Unit- 4

Object Oriented Programming

Object Oriented Programming Concepts

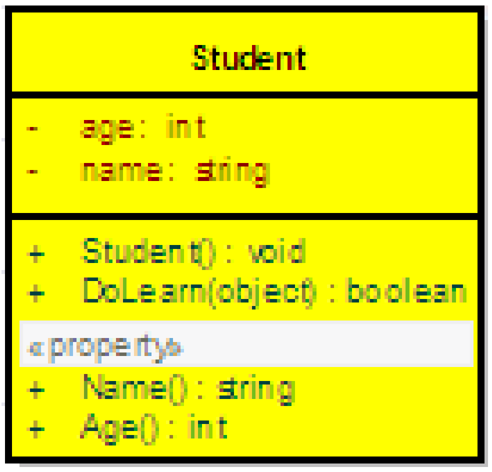
OOP is a design philosophy. It stands for Object Oriented Programming. Object-Oriented Programming (OOP) uses a different set of programming languages than old procedural programming languages (C, Pascal, etc.). Everything in OOP is grouped as self sustainable "objects". Hence, you gain re-usability by means of four main object-oriented programming concepts.

Concept of Class

A class is simply an abstract model used to define new data types. A class may contain any combination of encapsulated data (fields or member variables), operations that can be performed on the data (methods) and accessors to data (properties). For example, there is a class String in the System namespace of .Net Framework Class Library (FCL). This class contains an array of characters (data) and provide different operations (methods) that can be applied to its data like *ToLowerCase()*, *Trim()*, *Substring()*, etc. It also has some properties like *Length* (used to find the length of the string). A class in VB.Net is declared using the keyword Class and its members are enclosed with the *End Class* marker Class TestClass ' fields, operations and properties go here End Class Where *TestClass* is the name of the class or new data type that we are defining here.

Object An object can be considered a "thing" that can perform a set of related activities. The set of activities that the object performs defines the object's behavior. For example, the hand can grip something or a Student (object) can give the name or address. In pure OOP terms an object is an instance of a class.

Class A class is simply a representation of a type of object. It is the blueprint/ plan/ template that describe the details of an object. A class is the blueprint from which the individual objects are created. Class is composed of three things: a name, attributes, and operations.



```
public class Student
{
}
```

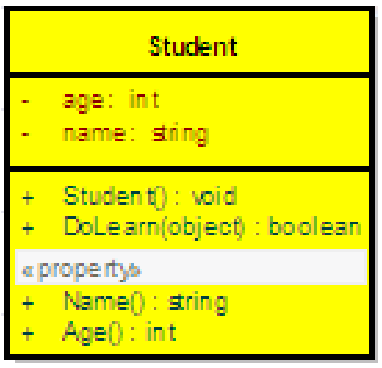
According to the sample given below we can say that the student object, named objectStudent, has created out of theStudent class.

Student objectStudent = new Student();

In real world, you'll often find many individual objects all of the same kind. As an example, there may be thousands of other bicycles in existence, all of the same make and model. Each bicycle has built from the same blueprint. In object-oriented terms, we say that the bicycle is an instance of the class of objects known as bicycles. In the software world, though you may not have realized it, you have already used classes. For example, the TextBoxcontrol, you always used, is made out of the TextBox class, which defines its appearance and capabilities. Each time you drag a TextBox control, you are actually creating a new instance of the TextBox class.

Encapsulation

The encapsulation is the inclusion within a program object of all the resources need for the object to function - basically, the methods and the data. In OOP the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties. The class is kind of a container or capsule or a cell, which encapsulate the set of methods, attribute and properties to provide its indented functionalities to other classes. In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system. That idea of encapsulation is to hide how a class does it but to allow requesting what to do.



In order to modularize/ define the functionality of a one class, that class can uses functions/ properties exposed by another class in many different ways. According to Object Oriented Programming there are several techniques, classes can use to link with each other and they are named association, aggregation, and composition.

There are several other ways that an encapsulation can be used, as an example we can take the usage of an interface. The interface can be used to hide the information of an implemented class.

IStudent myStudent = new LocalStudent();

IStudent myStudent = new ForeignStudent();

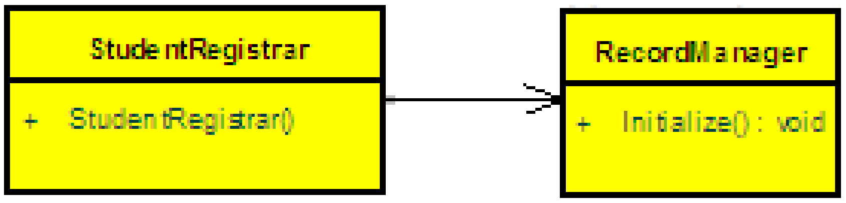
According to the sample above (let's assume that LocalStudent and ForeignStudent are implemented by the IStudentinterface) we can see how LocalStudent and ForeignStudent are hiding their, localize implementing information through the Student interface.

Association

Association is a (*a*) relationship between two classes. It allows one object instance to cause another to perform an action on its behalf. Association is the more general term that define the relationship between two classes, where as the aggregation and composition are relatively special.

```
public class StudentRegistrar
{
    public StudentRegistrar ();
        {
                new RecordManager().Initialize();
        }
}
```

In this case we can say that there is an association between StudentRegistrar and RecordManager or there is a directional association from StudentRegistrar to RecordManager or StudentRegistrar use a (*Use*) RecordManager. Since a direction is explicitly specified, in this case the controller class is the StudentRegistrar.



Interface

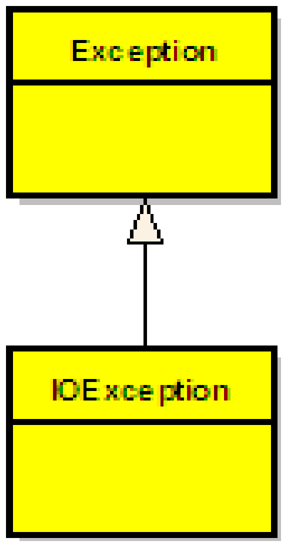
The Interface separates the implementation and defines the structure, and this concept is very useful in cases where you need the implementation to be interchangeable. Apart from that an interface is very

useful when the implementation changes frequently. Some say you should define all classes in terms of interfaces, but I think recommendation seems a bit extreme.
Interface can be used to define a generic template and then one or more abstract classes to define partial implementations of the interface. Interfaces just specify the method declaration (implicitly public and abstract) and can contain properties (which are also implicitly public and abstract). Interface definition begins with the keyword interface. An interface like that of an abstract class cannot be instantiated.
If a class that implements an interface does not define all the methods of the interface, then it must be declared abstract and the method definitions must be provided by the subclass that extends the abstract class. In addition to this an interfaces can inherit other interfaces.
The sample below will provide an interface for our LoggerBase abstract class.

```
public interface ILogger
{
bool IsThisLogError { get; }
}
```

Inheritance

Ability of a new class to be created, from an existing class by extending it, is called inheritance.



```
public class Exception
{
}
public class IOException : Exception
{
}
```

According to the above example the new class (IOException), which is called the derived class or subclass, inherits the members of an existing class (Exception), which is called the base class or super-class. The class IOException can extend the functionality of the class Exception by adding new types and methods and by overriding existing ones.
Just like abstraction is closely related with generalization, the inheritance is closely related with specialization. It is important to discuss those two concepts together with generalization to better understand and to reduce the complexity.
One of the most important relationships among objects in the real world is specialization, which can be described as the ―is-a‖ relationship. When we say that a dog is a mammal, we mean that the dog is a specialized kind of mammal. It has all the characteristics of any mammal (it bears live young, has hair), but it specializes these characteristics to the familiar characteristics of canis domesticus. A cat is also a mammal. As such, we expect it to share certain characteristics with the dog that are generalized in Mammal, but to differ in those characteristics that are specialized in cats.
The specialization and generalization relationships are both reciprocal and hierarchical. Specialization is just the other side of the generalization coin: Mammal generalizes what is common between dogs and cats, and dogs and cats specialize mammals to their own specific subtypes.
Similarly, as an example you can say that both IOException and SecurityException are of type Exception. They have all characteristics and behaviors of an Exception, That mean the IOException is a specialized kind of Exception. ASecurityException is also an Exception. As such, we expect it to share certain characteristic with IOException that are generalized in Exception, but to differ in those characteristics that are specialized in SecurityExceptions. In other words, Exception generalizes the shared

characteristics of both IOException and SecurityException, while IOException andSecurityException specialize with their characteristics and behaviors.
In OOP, the specialization relationship is implemented using the principle called inheritance. This is the most common and most natural and widely accepted way of implement this relationship.

Polymorphisms
Polymorphism is a generic term that means 'many shapes'. More precisely Polymorphism means the ability to request that the same operations be performed by a wide range of different types of things.
In OOP the polymorphisms is achieved by using many different techniques named method overloading, operator overloading and method overriding,

Overloading
The method overloading is the ability to define several methods all with the same name.

```
public class MyLogger
{
public void LogError(Exception e)
{
        // Implementation goes here
}
public bool LogError(Exception e, string }
message)
{
        // Implementation goes here
}
```

Operator Overloading
The operator overloading (less commonly known as ad-hoc polymorphisms) is a specific case of polymorphisms in which some or all of operators like +, - or == are treated as polymorphic functions and as such have different behaviors depending on the types of its arguments.

```
public class Complex
        {
                private int real;
                public int Real
                {get { return real; } }
                private int imaginary;
                public int Imaginary
                { get { return imaginary; } }
                public Complex(int real, int imaginary)
        {
                this.real = real;
                this.imaginary = imaginary;
        }
                public static Complex operator +(Complex c1, Complex c2)        }
                {
                return new Complex(c1.Real + c2.Real, c1.Imaginary +       }
                c2.Imaginary);
```

I above example I have overloaded the plus operator for adding two complex numbers. There the two properties named Real and Imaginary has been declared exposing only the required ―get‖ method, while the object's constructor is demanding for mandatory real and imaginary values with the user defined constructor of the class.

Methods - Sub Procedures and Functions

Methods are operations that can be performed on data. A method may take some input values through its parameters and may return a value of a particular data type. There are two types of methods in VB.Net: Sub Procedures and Functions.

Sub Procedure

A sub procedure is a method that does not return any values. A sub procedure in VB.Net is defined using the Sub keyword. A sub procedure may or may not accept parameters. A method that does not take any parameters is called a parameterless method. For example, the following is a parameterless sub procedure

```
Sub ShowCurrentTime()
Label1.Text("The current time is: " & DateTime.Now)
End Sub
```

Unit-5

Web Applications

Introduction to ASP.NET

ASP.NET is the new offering for Web developers from the Microsoft .It is not simply the next-generation of ASP; in fact, it is a completely re-engineered and enhanced technology that offers much, much more than traditional ASP and can increase productivity significantly. Because it has evolved from ASP, ASP.NET looks very similar to its predecessor—but only at first sight. Some items look very familiar, and they remind us of ASP. But concepts like Web Forms, Web Services, or Server Controls gives ASP.NET the power to build real Web applications.

Active Server Pages (ASP)

Microsoft Active Server Pages (ASP) is a server-side scripting technology. ASP is a technology that Microsoft created to ease the development of interactive Web applications. With ASP you can use client-side scripts as well as server-side scripts. Maybe you want to validate user input or access a database. ASP provides solutions for transaction processing and managing session state. ASP is one of the most successful language used in web development.

Problems with Traditional ASP

There are many problems with ASP if you think of needs for Today's powerful Web applications.

- Interpreted and Loosely-Typed Code
  ASP scripting code is usually written in languages such as JScript or VBScript. The script-execution engine that Active Server Pages relies on interprets code line by line, every time the page is called. In addition, although variables are supported, they are all loosely typed as variants and bound to particular types only when the code is run. Both these factors impede performance, and late binding of types makes it harder to catch errors when you are writing code.

- Mixes layout (HTML) and logic (scripting code)
  ASP files frequently combine script code with HTML. This results in ASP scripts that are lengthy, difficult to read, and switch frequently between code and HTML. The interspersion of HTML with ASP code is particularly problematic for larger web applications, where content must be kept separate from business logic.

- Limited Development and Debugging Tools
  Microsoft Visual InterDev, Macromedia Visual UltraDev, and other tools have attempted to increase the productivity of ASP programmers by providing graphical development environments. However, these tools never achieved the ease of use or the level of acceptance achieved by Microsoft Windows application development tools, such as Visual Basic or Microsoft Access. ASP developers still rely heavily or exclusively on Notepad.
  Debugging is an unavoidable part of any software development process, and the debugging tools for ASP have been minimal. Most ASP programmers resort to embedding temporary Response. Write statements in their code to trace the progress of its execution.

- No real state management
  Session state is only maintained if the client browser supports cookies. Session state information can only be held by using the ASP Session object. And you have to implement additional code if you, for example, want to identify a user.

- Update files only when server is down
  If your Web application makes use of components, copying new files to your application should only be done when the Web server is stopped. Otherwise it is like pulling the rug from under your application's feet, because the components may be in use (and locked) and must be registered.

- Obscure Configuration Settings
  The configuration information for an ASP web application (such as session state and server timeouts) is stored in the IIS metabase. Because the metabase is stored in a proprietary format, it can only be modified on the server machine with utilities such as the Internet Service Manager. With limited support for programmatically manipulating or extracting these settings, it is often an arduous task to port an ASP application from one server to another.

Introducing ASP.NET
ASP.NET was developed in direct response to the problems that developers had with classic ASP. Since ASP is in such wide use, however, Microsoft ensured that ASP scripts execute without modification on a machine with the .NET Framework (the ASP engine, ASP.DLL, is not modified when installing the .NET Framework). Thus, IIS can house both ASP and ASP.NET scripts on the same machine.

Features of ASP.NET
- Separation of Code from HTML
  To make a clean sweep, with ASP.NET you have the ability to completely separate layout and business logic. This makes it much easier for teams of programmers and designers to collaborate efficiently. This makes it much easier for teams of programmers and designers to collaborate efficiently.

- Support for compiled languages
  Developer can use VB.NET and access features such as strong typing and object-oriented programming. Using compiled languages also means that ASP.NET pages do not suffer the performance penalties associated with interpreted code. ASP.NET pages are precompiled to byte-code and Just In Time (JIT) compiled when first requested. Subsequent requests are directed to the fully compiled code, which is cached until the source changes.

- Use services provided by the .NET Framework
  The .NET Framework provides class libraries that can be used by your application. Some of the key classes help you with input/output, access to operating system services, data access, or even debugging. We will go into more detail on some of them in this module.

- Graphical Development Environment
  Visual Studio .NET provides a very rich development environment for Web developers. You can drag and drop controls and set properties the way you do in

Visual Basic 6. And you have full IntelliSense support, not only for your code, but also for HTML and XML.

- State management
  To refer to the problems mentioned before, ASP.NET provides solutions for session and application state management. State information can, for example, be kept in memory or stored in a database. It can be shared across Web farms, and state information can be recovered, even if the server fails or the connection breaks down.

- Update files while the server is running
  Components of your application can be updated while the server is online and clients are connected. The Framework will use the new files as soon as they are copied to the application. Removed or old files that are still in use are kept in memory until the clients have finished.

- XML-Based Configuration Files
  Configuration settings in ASP.NET are stored in XML files that you can easily read and edit. You can also easily copy these to another server, along with the other files that comprise your application.

File name extensions
Web applications written with ASP.NET will consist of many files with different file name extensions. The most common are listed here. Native ASP.NET files by default have the extension .aspx (which is, of course, an extension to .asp) or .ascx. Web Services normally have the extension .asmx.
Your file names containing the business logic will depend on the language you use. So, for example, a VB file would have the extension .aspx.vb & C# file would have the extension .aspx.cs.
Another one worth mentioning is the ASP.NET application file Global.asax - in the ASP world formerly known as Global.asa. But now there is also a code behind file Global.asax.vb, for example, if the file contains Visual Basic.NET code. Global.asax is an optional file that resides in the root directory of your application, and it contains global logic for your application.

Directives
You can use directives to specify optional settings used by the page compiler when processing ASP.NET files. For each directive you can set different attributes. One example is the language directive at the beginning of a page defining the default programming language.

Code Declaration Blocks
Code declaration blocks are lines of code enclosed in <script> tags. They contain the runat=server attribute, which tells ASP.NET that these controls can be accessed on the server and on the client. Optionally you can specify the language for the block. The code block itself consists of the definition of member variables and methods.

Code Render Blocks
Render blocks contain inline code or inline expressions enclosed by the character sequences shown here. The language used inside those blocks could be specified through a directive like the one shown before.

HTML Control Syntax
You can declare several standard HTML elements as HTML server controls. Use the element as you are familiar with in HTML and add the attribute runat=server. This causes the HTML element to be treated as a server control. It is now programmatically accessible by using a unique ID. HTML server controls must reside within a <form> section that also has the attribute runat=server.

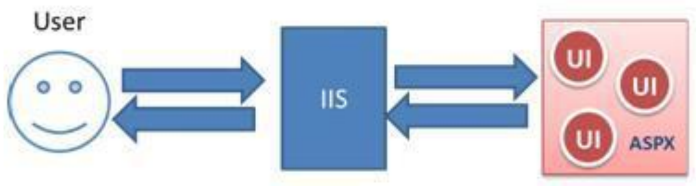
Custom Control Syntax
There are two different kinds of custom controls. On the one hand there are the controls that ship with .NET, and on the other hand you can create your own custom controls. Using custom server controls is the best way to encapsulate common programmatic functionality.
Just specify elements as you did with HTML elements, but add a tag prefix, which is an alias for the fully qualified namespace of the control. Again you must include the runat=server attribute. If you want to get programmatic access to the control, just add an Id attribute.
You can include properties for each server control to characterize its behavior. For example, you can set the maximum length of a TextBox. Those properties might have sub properties; you know this principle from HTML. Now you have the ability to specify, for example, the size and type of the font you use (font-size and font-type).
The last attribute is dedicated to event binding. This can be used to bind the control to a specific event. If you implement your own method MyClick, this method will be executed when the corresponding button is clicked if you use the server control event binding shown in the slide.

Data Binding Expression
You can create bindings between server controls and data sources. The data binding expression is enclosed by the character sequences <%# and %>. The data-binding model provided by ASP.NET is hierarchical. That means you can create bindings between server control properties and superior data sources.

Server-side Object Tags
If you need to create an instance of an object on the server, use server-side object tags. When the page is compiled, an instance of the specified object is created. To specify the object use the identifier attributes. You can declare (and instantiate) .NET objects using class as the identifier, and COM objects using either progid or classid.

Server-side Include Directives
With server-side include directives you can include raw contents of a file anywhere in your ASP.NET file. Specify the type of the path to filename with the pathtype attribute. Use either File, when specifying a relative path, or Virtual, when using a full virtual path.

What is Web Forms?
Microsoft first brought out ASP.NET Web Forms from ASP which solved lots of problems by creating higher level abstraction over stateless web and simulated stateful model for Web developers. In web forms concepts like self postback (post form data to same page) and ViewState (maintain control values during postbacks) are introduced. And the most interesting part is it’s not required to write even a single line of code. With Web Forms Microsoft tried to bring the Visual Basic model into web.



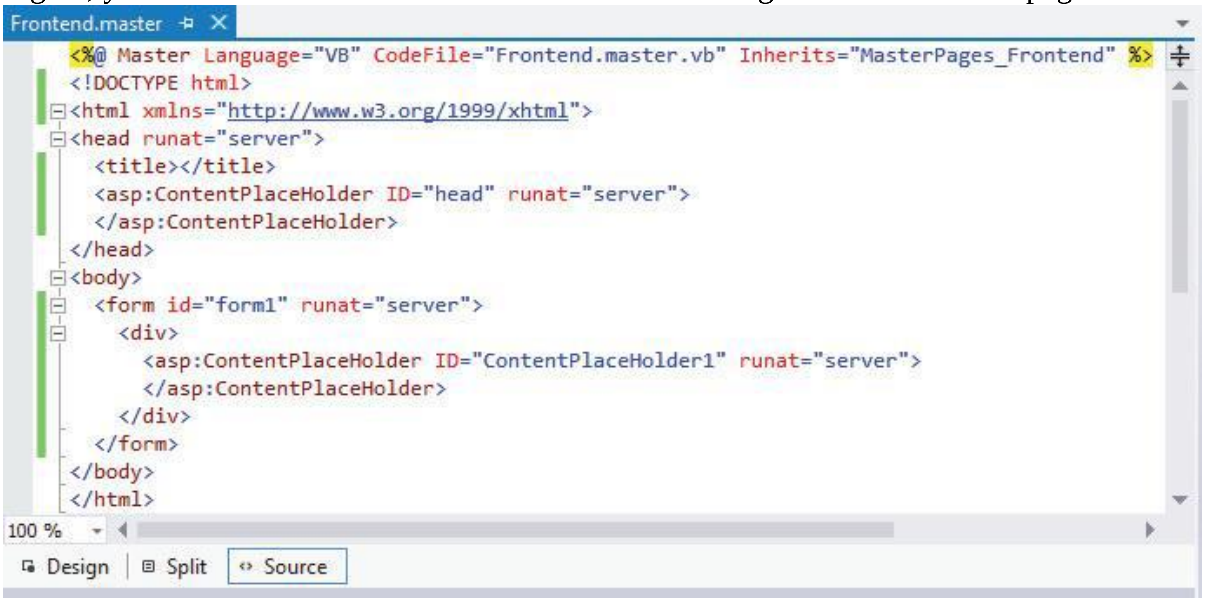
WORKING WITH WEB FORMS

Web Forms, represented by .aspx files, are the core of any ASP.NET 4.5 Web Forms website. They are the actual pages that users see in their browsers when they visit your site.

Web Forms can contain a mix of HTML, ASP.NET Server Controls, client-side JavaScript, CSS, and programming logic. To make it easier to see how all this code ends up in the browser, Visual Studio offers a number of different views on your pages.

The Different Views on Web Forms

Visual Studio enables you to look at your Web Form from a few different angles. When you have a file with markup—like a Web Form or master page—open in the Document Window, you see three buttons at the bottom-left corner of the window. With these buttons, visible in Figure, you can switch between the different views. This figure shows a master page



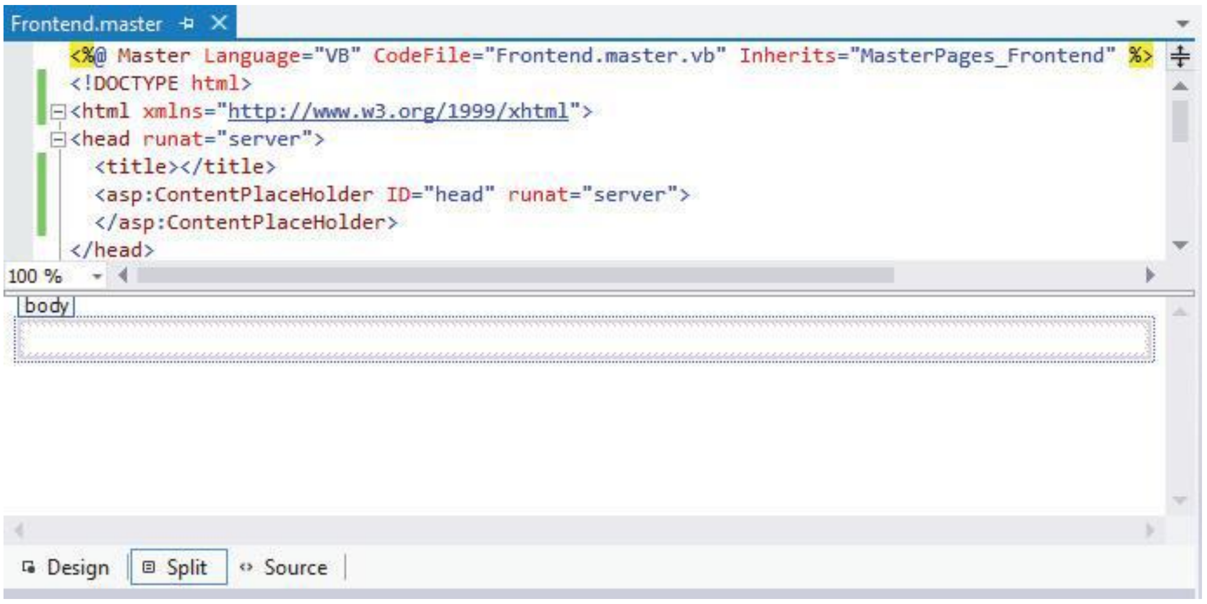
```
<%@ Master Language="VB" CodeFile="Frontend.master.vb" Inherits="MasterPages_Frontend" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
  <asp:ContentPlaceHolder ID="head" runat="server">
  </asp:ContentPlaceHolder>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
      </asp:ContentPlaceHolder>
    </div>
  </form>
</body>
</html>
```

Source View is the default view when you open a page. It shows you the raw HTML and other markup for the page, and is very useful if you want to tweak the contents of a page and you have a good idea of what you want to change and where. I use the term Markup View rather than Source View to refer to the markup of ASPX and HTML pages.

The Design button enables you to switch the Document Window into Design View, which gives you an idea of how the page will end up. When in Design View, you can use the Visual Aids and

Formatting Marks submenus from the main View menu to control visual markers like line breaks, borders, and spaces. Both submenus offer a menu item called Show that enables you to turn all the visual aids on or off at once. Turning both off is useful if you want to have an idea of how the page ends up in the browser. You should, however, use Design View only to get an idea of how the page will end up. Although Visual Studio has a great rendering engine that renders the page in Design View pretty well, you should always check your pages in different browsers as well, because what you see in Visual Studio is the markup for the page before it gets processed.

The Split button enables you to look at Design View and Markup View at the same time, as you can see in following figure. Split View is great if you want to see the code that Visual Studio generates when you add controls to the Design View of your page. The other way around is very useful too: When you make changes to the markup of the page in Markup View, you can see how it ends up in Design View. Sometimes Design View becomes out of sync with Markup View. If that's the case, a message appears at the top of Design View. Simply clicking the message or saving the entire page is enough to update the Design window.



```
<%@ Master Language="VB" CodeFile="Frontend.master.vb" Inherits="MasterPages_Frontend" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
  <asp:ContentPlaceHolder ID="head" runat="server">
  </asp:ContentPlaceHolder>
</head>
```

In addition to the HTML and other markup you see in the Markup View window, a Web Form can also contain code in Visual Basic .NET. Where this code is placed depends on the type of Web Form you create.

Choosing between Code behind and Pages with Inline Code

Web Forms come in two flavors: either as an .aspx file with a Code Behind file (a file named after the Web Form with an additional .vb extension) or as .aspx files that have their code embedded, often referred to as Web Forms with inline code. It's important to understand the difference between these types of Web Forms. At first, Web Forms with inline code seem a little easier to understand. Because the code needed to program your website is part of the very same Web Form, you can clearly see how the code relates to the file. However, as your page gets larger and you add more functionality to it, it's often easier if you have the code in a separate file. That way, it's completely separate from the markup, enabling you to focus on the task at hand.

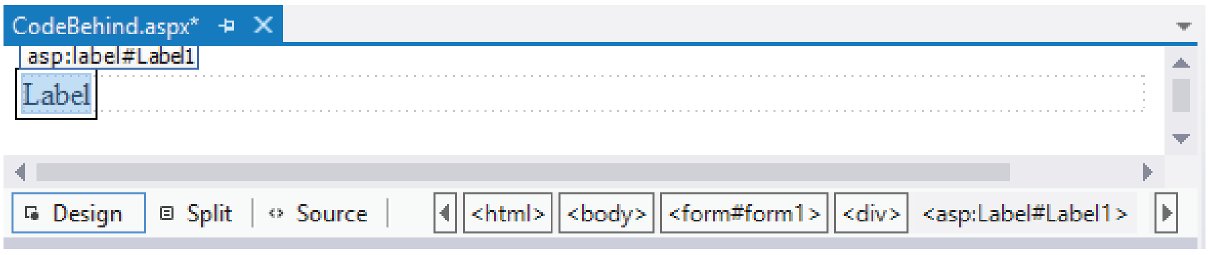
Adding Web Forms with Code to Your Site

To add web forms in the website project you need to do the following procedure.

1. In the Solution Explorer, right-click your website and choose Add New Item. In the dialog box that appears, choose your preferred programming language on the left, click the Web Form template, and name the file **CodeBehind.aspx**. Make sure that the check box for Place Code in Separate File is selected. Finally, click the Add button. The page should open in Markup View so you can see the HTML for the page.

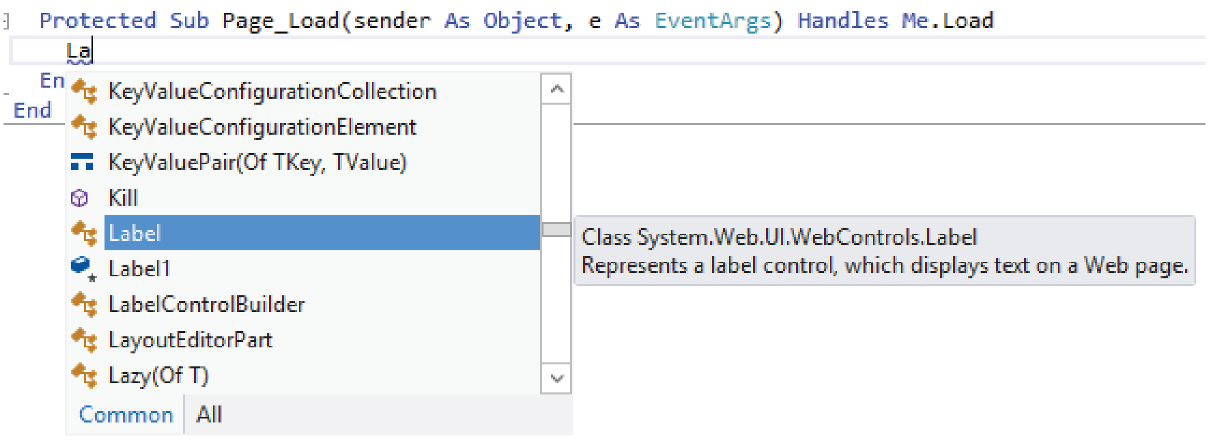
2. At the bottom of the Document Window, click the Design button to switch the page from Markup View into Design View. The page you see has a white background with a small,

dashed rectangle at the top of it. The dashed rectangle represents the <div> element you saw in Markup View.
3. From the Toolbox, drag a Label control from the Standard category and drop it in the dashed area of the page. Remember, you can open the Toolbox with the shortcut Ctrl+Alt+X if it isn't open yet.
In Design View, your screen should now look like following figure.



4. Double-click somewhere in the white area below the dashed line of the <div> element. Visual Studio switches from Design View into the Code Behind of the file and adds code that fires when the page loads in the browser
Following code will appear at the code window.
Protected Sub Page_Load(sender As Object, e As EventArgs) Handles Me.Load
End Sub

In most cases, Visual Studio adds it for you automatically, as you just saw now it's important to realize that the code you're going to place between the lines that start with Protected Sub and End Sub in Visual Basic will be run when the page is requested in the browser.
6. Place your cursor in the open line in the code that Visual Studio created and add the bolded line of code that assigns today's date and time to the label, which will eventually show up in the browser:
Protected Sub Page_Load(sender As Object, e As EventArgs) Handles Me.Load
Label1.Text = "Hello World; the time is now " & DateTime.Now.ToString()
End Sub
Note that as soon as you type the L for Label1, you get a list with options to choose from. This is part of **Visual Studio's IntelliSense**, a great tool that helps you rapidly write code. Instead of typing the whole word Label1, you simply type the letter L or the letters La and then you pick the appropriate item from the list, visible in following figure.

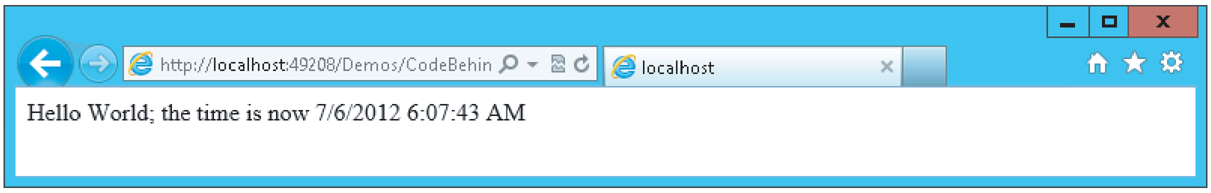


```
Protected Sub Page_Load(sender As Object, e As EventArgs) Handles Me.Load
    La
```

KeyValueConfigurationCollection
KeyValueConfigurationElement
KeyValuePair(Of TKey, TValue)
Kill
Label
Label1
LabelControlBuilder
LayoutEditorPart
Lazy(Of T)
Common
All

Class System.Web.UI.WebControls.Label
Represents a label control, which displays text on a Web page.

To complete the selected word, you can press Enter or Tab or even the period. In the latter case, you immediately get another list that enables you to pick the word Text simply by typing the first few letters, completing the word by pressing the Tab or Enter key. This feature is a real productivity tool because you can write code with a minimum of keystrokes.
7. Right-click the CodeBehind.aspx page in the Solution Explorer and choose View in Browser. Depending on the default browser you've configured for your computer, the browser name in the parentheses may be different.
8. Click Yes if you get a dialog box that asks if you want to save the changes, and then the page will appear in the browser, similar to the browser window you see in following figure.

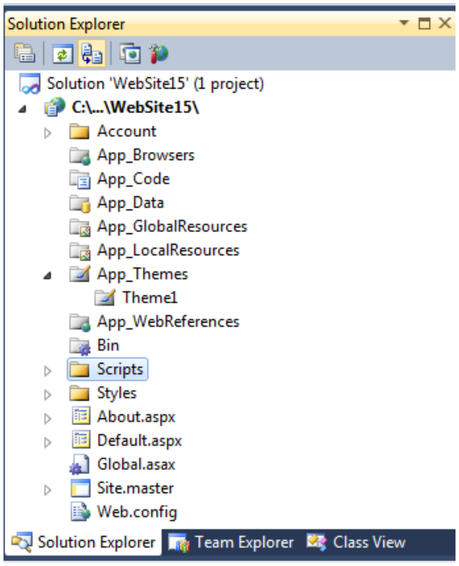


Hello World; the time is now 7/6/2012 6:07:43 AM

ASP.NET Directory Structure When using ASP.NET to create web application, you have an ASP.NET directory structure which can be determined by you the developer's choices. There are a few reserved directory names which cannot be used but a site can have as much as directories as required.
The ASP.NET Directory structure is typically reflected directly in the URL, and even though ASP.NET provides a way to intercept the request at a point during web page processing, it is not forced to funnel requests through a central application or front controller.
The following are some of the Special ASP.NET Directory names available in ASP.NET:

App_Code
Is the directory used for any code used in the ASP.NET web application. The ASP.NET server will automatically compile these files and any subdirectories within the App_Code folder into an assembly which can be then accessed by the code in any page of the ASP.NET website.
The App_Code will be used for:
 Data access abstraction code
 Model code
 Business code
 Any site specific http handlers and modules
 Any Web service implementations

Note: An alternative to using App_Code the developer may opt to provide a separate assembly with pre-compiled code.



App_Data
The App_Data ASP.NET Directory is the default directory for any database used by the ASP.NET Website. These databases might include Access (mdb) files or SQL Server (mdf) files. The App_Data is the only directory with Write Access enabled for the ASP.NET web application.
App_LocalResources
The ASP.NET App_LocalResources Directory holds any localized resources used by the ASP.NET web application. These may includes different version of the same page for instance a page with the same content but in two different languages.

App_GlobalResources
The ASP.NET App_GlobalResources Directory is used to hold any ―resx‖ files with localized resources which can be used at any ASP.NET web page in the web site. The ASP.NET App_Global Resources folder is where the ASP.NET developer uses to store any localized messages which might be used on more than one ASP.NET web page.
App_Themes
The ASP.NET App_Themes Directory is used to contain any files related to ASP.NET themes. ASP.NET themes can be used to make a consistent appearance throughout the whole ASP.NET Website and makes it easier to change the appearance of the website.
App_WebReferences
The ASP.NET App_WebReferences is used to contain any discovery files and WSDL files for references to Web services to be used in the ASP.NET website.
Bin
The ASP.NET Bin Directory is used to hold any compiled code (.dll files). This compiled code can be used as reference in your application for

- Controls
- Components
- Other codes of you application

Any classes represented by code in the Bin folder are automatically referenced in your application.

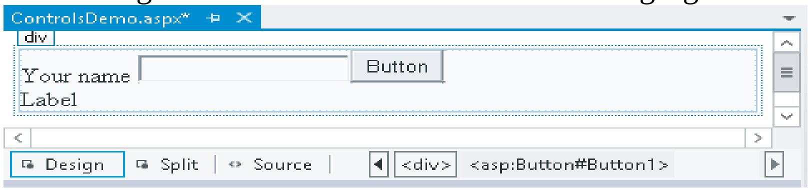
Working with Web form Controls
Almost all the Web Forms pages you build in Visual Studio will contain one or more controls. These controls come in all sorts and sizes, ranging from simple controls like a Button and a Label to complex controls like the TreeView and the ListView that are capable of displaying data from a data source (like a database or an XML file).
The architecture of ASP.NET Controls is deeply integrated into ASP.NET, giving the controls a feature set that is quite unique in today's technologies for building websites. The section that follows gives you a thorough look at the many controls that are available in the Visual Studio Toolbox.

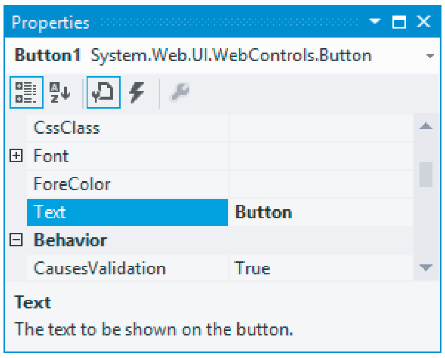
Introduction to Web Controls
It's important to understand how web controls operate and how they are completely different from the way you define controls in other languages like classic ASP or PHP (another popular programming language for creating dynamic websites). In ASP.NET, the controls ―live‖ on the server inside an ASPX page. When the page is requested in the browser, the server-side controls are processed by the ASP.NETrun time. The controls then emit client-side HTML code that is appended to the final page output. It's this HTML code that eventually ends up in the browser, where it's used to build up the page. So, instead of defining HTML controls in your pages directly, you define an ASP.NET Server Control with the following syntax, where the italicized parts differ for each control:
<asp:TypeOfControl ID="ControlName" runat="server" />
For the controls that ship with ASP.NET 4.5 you always use the asp: prefix followed by the name of the control. For example, to create a TextBox that can hold the same welcome message and current time, you can use the following syntax:
<asp:TextBox ID="Message" runat="server" />
Note that the control has two attributes: ID and runat. The ID attribute is used to uniquely identify a control on the page, so you can program against it. It's important that each control on the page has a unique ID; otherwise the ASP.NET run time won't understand what control

you're referring to. If you accidentally type a duplicate control ID, Visual Studio signals the problem in the error list. The mandatory runat attribute is used to indicate that this is a control that lives on the server. Without this attribute, the controls won't be processed and will end up directly in the HTML source. If you ever feel you're missing a control in the final output in the browser, ensure that the control has this required attribute. You can easily add the runat attribute to an existing element using a code snippet by typing runat and pressing the Tab key.
The preceding example of the TextBox uses a self-closing tag where the closing slash (/) is embedded in the opening tag. This is quite common for controls that don't need to contain child content such as text or other controls. However, the long version, using a separate closing tag, is acceptable as well: *<asp:TextBox ID="Message" runat="server"></asp:TextBox>* You can program against this text box from code that is either placed inline with the page or in a separate Code Behind file. To set the welcome message and the time, you can use the following code: *Message.Text = "Hello World, the time is " & DateTime.Now.TimeOfDay.ToString()* The definition of the control in the markup section of the page is now separated from the actual code that defines the text displayed in the text box, making it easier to define and program the text box (or any other control) because it enables you to focus on one task at a time. You can either declare the control and its visual appearance in the markup section of the page, or program its behavior from a code block. In general, controls defined in Markup View are not case-sensitive, although some of the values you can set are case-sensitive. **Working with Server Controls** Add a TextBox, a Label, and a Button control to a page. When you request the page in the browser, these server controls are transformed into HTML, that is then sent to the client. By looking at the final HTML for the page in the browser, you'll see how the HTML is completely different from the initial ASP.NET markup.
1. Open new web project in Visual Studio.

2. In the Solution Explorer, create a new Web Form called ControlsDemo.aspx. Choose your programming language and make sure the Web Form uses Code Behind.

3. Switch to Design View. From the Standard category of the Toolbox, drag a TextBox, a Button, and a Label control onto the design surface within the dashed lines of the <div> tag that was added for you when you created the page.

4. Type the text Your name in front of the TextBox and add a line break between the Button and the Label by positioning your cursor between the two controls in Design View and then pressing Enter.
Your Design View should now look like following figure.


5. Right-click the Button control and choose Properties to open up the Properties Grid for the control. The window that appears, will be as shown in following figure, enables you to change the properties for the control, which in turn influences the way the control looks and behaves at run time.

Properties
Button1 System.Web.UI.WebControls.Button
CssClass
Font
ForeColor
Text
Button
Behavior
CausesValidation
True
Text
The text to be shown on the button.
6. Set the control's Text property to Submit Information and set its ID (which you'll find all
the way down at the bottom of the list wrapped in parentheses) to SubmitButton.

7. Change the ID of the TextBox to YourName using the Properties Grid.
8. Clear the Text property of the Label using the Properties Grid. You can right-click the property’s label in the grid and choose Reset. Set its ID to Result.
9. Still in Design View, double-click the Button control to have Visual Studio add some code to the Code Behind of the page that will be fired when the button is clicked in the browser. Add the following line of code to the code block:

```
Protected Sub SubmitButton_Click(sender As Object,e As EventArgs) Handles
SubmitButton.Click
Result.Text = "Your name is " & YourName.Text
End Sub
```

10. Save the changes to the page and then open it in the browser by pressing Ctrl+F5. Don’t click the button yet, but open up the source of the page by right-clicking the page in the browser and choosing View Source or View Page Source. You should see the following HTML code (I changed the formatting slightly so the HTML fits on the page):

```
<div>
Your name <input name="YourName" type="text" id="YourName" />
<input type="submit" name="SubmitButton" value="Submit Information"
id="SubmitButton" />
<br />
<span id="Result"></span>
</div>
```

11. Switch back to your browser, fill in your name in the text box, and click the button. When the page is done reloading, open up the source for the page in the browser again using the browser’s rightclick menu. The code should now look like this:

```
<div>
Your name <input name="YourName" type="text" value="John" id="YourName" />
<input type="submit" name="SubmitButton" value="Submit Information"
id="SubmitButton" />
<br />
<span id="Result">Your name is John</span>
</div>
```

HTML Controls for ASP.NET Web Pages

By default, HTML elements within an ASP.NET file are treated as literal text and you cannot reference them in server-side code. To make these elements programmatically accessible, you can indicate that an HTML element should be treated as a server control by adding the runat="server" attribute. You can also set the element's id attribute to give you way to programmatically reference the control. You then set attributes to declare property arguments and event bindings on server control instances.
Besides working with server-side controls (HTML or WEB) every asp.net page will generally contain pure (or native) HTML elements like tables, lists, images, etc.

"Pure" HTML - tags, elements, attributes,...

HTML elements are structural parts of a Web Form, defined by HTML tags and tag attributes. By default, classic HTML elements within an ASP.NET web form are treated as literal text and are programmatically inaccessible to page developers - such elements are client-side operated - the client's browser uses the tags to render the page.
For example, This is a link is a hyperlink element; <a> is a tag, href="..." is an attribute. Another example: <input type=button> - will render a button element on a web form. In order

to programmatically operate on a HTML element from your page's code-behind class, you'll need to server-side "enable" an element you want to operate on. HTML elements that are marked "server-side" are called HTML Controls. HTML controls mimic the actual HTML elements you would use if you were using Notepad, HTML Kit, FrontPage or any other HTML editor to draw your UI.

ASP.NET Events & Event Handlers
Event is an action or occurrence like mouse click, key press, mouse movements, or any system generated notification. The processes communicate through events. For example, Interrupts are system generated events. When events occur the application should be able to respond to it.
In ASP.Net an event is raised on the client, and handled in the server. For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server.
The server has a subroutine describing what to do when the event is raised; it is called the event-handler. Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler, and if it has, the event handler is executed

Event Arguments:
ASP.Net event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is called the event argument.
The general syntax of an event is:
private void EventName (object sender, EventArgs e);
Application and Session Events:
The most important application events are:
- Application_Start . it is raised when the application/website is started
- Application_End . it is raised when the application/website is stopped

Similarly, the most used Session events are:
- Session_Start . it is raised when a user first requests a page from the application
- Session_End . it is raised when the session ends

Page and Control Events:
Common page and control events are:
- **DataBinding** . raised when a control bind to a data source
- **Disposed** . when the page or the control is released
- **Error** . it is an page event, occurs when an unhandled exception is thrown
- **Init .** raised when the page or the control is initialized
- **Load** . raised when the page or a control is loaded
- **PreRender** . raised when the page or the control is to be rendered
- **Unload** . raised when the page or control is unloaded from memory

Event Handling Using Controls:
All ASP.Net controls are implemented as classes, and they have events which are fired when user performs certain action on them. For example, when a user clicks a button the 'Click' event is generated. For handling these events there are in-built attributes and event handlers. To respond to an event, the event handler is coded.

By default Visual Studio creates an event handler by including a Handles clause on the Sub procedure. This clause names the control and event that the procedure handles.
The common control events are:

Event	Attribute	Controls
Click	OnClick	Button, image button, link button, image map
Command	OnCommand	Button, image button, link button
TextChanged	OnTextChanged	Text box
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list, list box, radio button list, check box list.
CheckedChanged	OnCheckedChanged	Check box, radio button

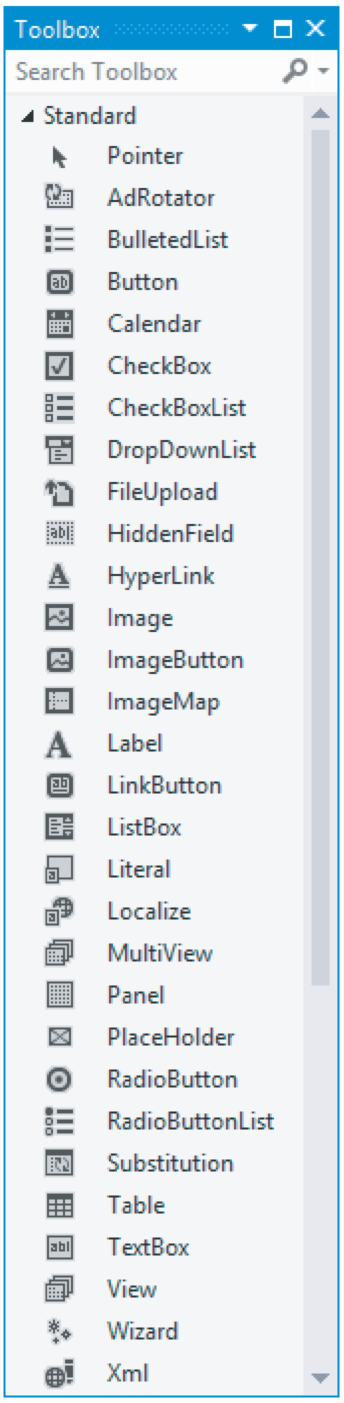
Some events cause the form to be posted back to the server immediately; these are called the postback events. For example, the click events like, Button.Click. Some events are not posted back to the server immediately; these are called non-postback events.
For example, the change events or selection events, such as, TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.

. **Default Events:**
The default event for the Page object is the Load event. Similarly every control has a default event. For example, default event for the button control is the Click event.
The default event handler could be created in Visual Studio, just by double clicking the control in design view. The following table shows some of the default events for common controls:

Control	Default Event
AdRotator	AdCreated
BulletedList	Click
Button	Click
Calender	SelectionChanged
CheckBox	CheckedChanged
CheckBoxList	SelectedIndexChanged
DataGrid	SelectedIndexChanged
DataList	SelectedIndexChanged
DropDownList	SelectedIndexChanged
HyperLink	Click
ImageButton	Click
ImageMap	Click
LinkButton	Click
ListBox	SelectedIndexChanged
Menu	MenuItemClick
RadioButton	CheckedChanged
RadioButtonList	SelectedIndexChanged

TYPES OF CONTROLS

ASP.NET 4.5 comes with a large number of server controls, supporting most of your web development needs. To make it easy for you to find the right controls, they have been placed in separate control categories in the Visual Studio Toolbox (accessible by pressing Ctrl+Alt+X). figure shows the Toolbox with all the available categories. In the following sections, you see the controls in each category and the tasks for which they are designed.



Standard Controls

The Standard category contains many of the basic controls that almost any web page needs like the *TextBox*, *Button*, and *Label* controls. Following figure shows all the controls in the Standard category. Many of the controls probably speak for themselves, so instead of giving you a detailed description of them all, the following sections briefly highlight a few important ones.

Simple Controls

The Toolbox contains a number of simple and straightforward controls, including *TextBox*, *Button*, *Label*, *HyperLink*, *RadioButton*, and *CheckBox*. Their icons in the Toolbox give you a good clue as to how they end up in the browser. In the remainder of this book, you see these controls used many times. In ASP.NET 4.5 the *TextMode* property of the *TextBox* control has been expanded to support new HTML5 types such as *DateTime*, *Email*, and *Number*.

List Controls

The standard category also contains a number of controls that present themselves as lists in the browser. These controls include *ListBox*, *DropDownList*, *CheckBoxList*, *RadioButtonList*, and *BulletedList*. To add items to the list, you define *<asp:ListItem>* elements between the opening and closing tags of the control, as shown in the following example:

```
<asp:DropDownList ID="FavoriteLanguage" runat="server">
<asp:ListItem Value="C#">C#</asp:ListItem>
<asp:ListItem Value="Visual Basic">Visual
Basic</asp:ListItem>
<asp:ListItem Value="CSS">CSS</asp:ListItem>
</asp:DropDownList>
```

The *DropDownList* enables a user to select only one item at a time. To see the currently active and selected item of a list control programmatically, you can look at its *SelectedValue*, *SelectedItem*,or *SelectedIndex* properties. *SelectedValue* returns a string that contains the value for the selected item, like C# or Visual Basic in the preceding example.

SelectedIndex returns the zero-based index of the item in the list. With the preceding example, if the user had chosen C#, *SelectedIndex* would be 0. Similarly, if the user had chosen CSS, the index would be 2 (the third item in the list). For controls that allow multiple selections (like *CheckBoxList* and *ListBox*), you can loop through the Items collection and see what items are selected. In this case, *SelectedItem* returns only the first selected item in the list; not all of them. Note that in the browser, both the *DropDownList* and the *ListBox* control render as a <*select*> element. Attributes such as size and multiple set by these two controls determine the appearance and behavior of the HTML element in the browser. The *BulletedList* control doesn't allow a user to make selections, and as such doesn't support these properties. To see how to add list items to your list control, and how to read the selected values, the following example guides you through creating a simple Web Form with two list controls that ask users for their favorite programming language.

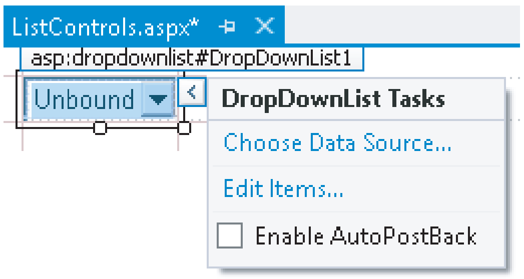
Working with List Controls In this example you add *two list controls* to a page. Additionally, you add a *button* that, when clicked, displays the selected items as text in a *Label control*.

1. In the main folder, create a new *Web Form* called *ListControls.aspx*. Make sure you create a Code Behind file by checking the Place Code in Separate File option.

2. Switch to Design View and drag a *DropDownList* from the Toolbox onto the design surface of the page within the dashed border of the <div> element that is already present in your page.

3. Notice that as soon as you drop the *DropDownList* control on the page, a pop-up menu appears that is labeled

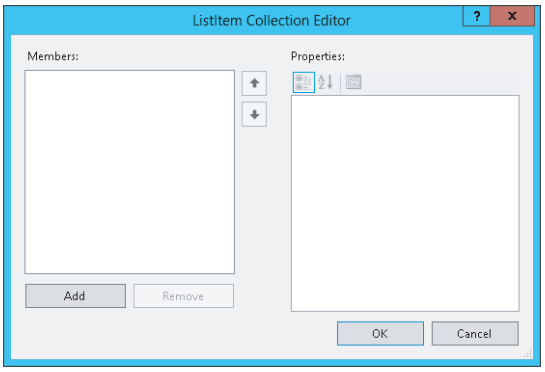
DropDownList Tasks, as shown in following figure



4. This pop-up menu is called the Smart Tasks panel. When it appears, it gives you access to the most common tasks of the control it belongs to. In the case of the *DropDownList*, you get three options. The first option enables you to bind the control to a data source, the second item enables you to manually add items to the list, and the last option sets the *AutoPostBack* property of the control. With this option checked, the control submits the page in which it is contained back to the server as soon as the user chooses a new item from the list.

The Smart Tasks panel appears only for the more complex controls that have a lot of features. You won't see it for simple controls like *Button* or *Label*. To reopen the Smart Tasks panel, right-click the control in the designer and choose Show Smart Tag. Alternatively, click the small arrow at the top-right corner of the control, visible in Figure, or press Shift+Alt+F10 when the control is selected. You can also open the Smart Tasks panel from markup view. Simply click anywhere on the opening or closing tag of a control or other piece of markup and press Ctrl+Dot
(Ctrl+.). Alternatively, hover over the tiny blue rectangle at the start of the opening tag and then click the grey arrow that appears.

On the Smart Tasks panel of the *DropDownList*, click the Edit Items link to bring up the *ListItem* Collection Editor, shown in following figure.



This dialog box enables you to add new items to the list control. The items you add through this window are added as <*asp:ListItem*> elements between the tags for the control.

5. Click the Add button on the left side of the screen to insert a new list item. Then in the Properties Grid on the right, enter **C#** for the

Text property and press Tab. As soon as you tab away from the *Text property*, the value is copied to the Value property as well. This is convenient if you want both the Text and the Value property to be the same. However, it's perfectly OK (and quite common) to assign a different value to the *Value property*.

6. Repeat step 5 twice, this time creating list items for **Visual Basic** and **CSS**. You can use the up and down arrow buttons in the middle of the dialog box to change the order of the items in the list.

7. Finally, click OK to insert the items in the page. You should end up with the following code in Markup View:

```
<asp:DropDownList ID="DropDownList1" runat="server">
<asp:ListItem>C#</asp:ListItem>
<asp:ListItem>Visual Basic</asp:ListItem>
<asp:ListItem>CSS</asp:ListItem>
</asp:DropDownList>
```

8. In Markup View drag a *CheckBoxList* control from the Toolbox directly into the code window, right after the *DropDownList.*

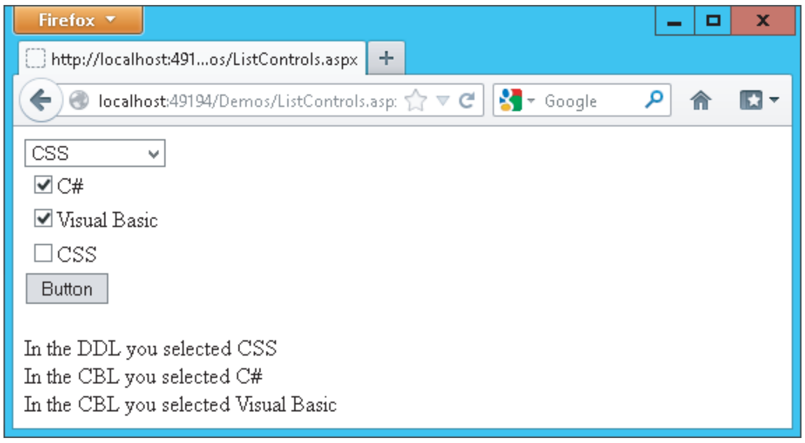
9. Copy the three *<asp:ListItem>* elements from the *DropDownList* you created in steps 5 and 6 and paste them between the opening and closing tags of the *CheckBoxList.* You should end up with this code:

```
<asp:ListItem>CSS</asp:ListItem>
</asp:DropDownList>
<asp:CheckBoxList ID="CheckBoxList1" runat="server">
<asp:ListItem>C#</asp:ListItem>
<asp:ListItem>Visual Basic</asp:ListItem>
<asp:ListItem>CSS</asp:ListItem>
</asp:CheckBoxList>
```

10. Switch to Design View and drag a Button from the Toolbox in Design View to the right of the *CheckBoxList* control. The Button will be placed below the *CheckBoxList*. Next, drag a *Label* control and drop it to the right of the Button. Create some room between the *Button* and the *Label* by positioning your cursor between the controls and then pressing Enter twice. Double-click the *Button* to open the Code Behind of the page. In the code block, add the following bolded code, which will be executed when the user clicks the button:

```
Protected Sub Button1_Click(sender As Object, e As EventArgs) _
Handles Button1.Click
Label1.Text = "In the DDL you selected " &
DropDownList1.SelectedValue & "<br />"
For Each item As ListItem In CheckBoxList1.Items
If item.Selected = True Then
Label1.Text &= "In the CBL you selected " & item.Value & "<br />"
End If
Next
End Sub
```

11. Save the changes to the page and then request it in the browser. Choose an item from the *DropDownList,* check one or more items in the *CheckBoxList,* and click the button. You should see something similar to following figure.


Firefox
http://localhost:491...os/ListControls.aspx
localhost:49194/Demos/ListControls.asp:
Google
CSS
C#
Visual Basic
CSS
Button

In the DDL you selected CSS
In the CBL you selected C#
In the CBL you selected Visual Basic

Other Standard Controls
This section briefly discusses the remainder of the controls in the Standard category of the Toolbox.

LinkButton and ImageButton
The *LinkButton* and the *ImageButton* controls operate similarly to an ordinary *Button* control. Both of them cause a *postback* to the server when they are clicked. The *LinkButton* presents itself as a simple <*a*> element, but posts back (using JavaScript) instead of requesting a new page. The *ImageButton* does the same, but displays an image that the user can click to trigger the *postback.*

Image and ImageMap
These controls are pretty similar in that they display an image in the browser. The *ImageMap* enables you to def ne *hotspots* on the image that, when clicked, either cause a *postback* to the server or navigate to a different page.

FileUpload
The *FileUpload* control enables a user to upload files that can be stored on the server.
Literal, Localize, and Substitute
All three controls look a little like the Label control because they can all display static text or HTML. The biggest advantage of the Literal is that it renders no additional tag itself; it displays only what you assign to its Text property, and is thus very useful to display HTML or JavaScript that you build up in the Code Behind or that you retrieve from a database.
The *Localize* control is used in multilingual websites and is able to retrieve its contents from translated resource files. The Substitute control is used in advanced caching scenarios and enables you to update only parts of a page that is otherwise cached completely.

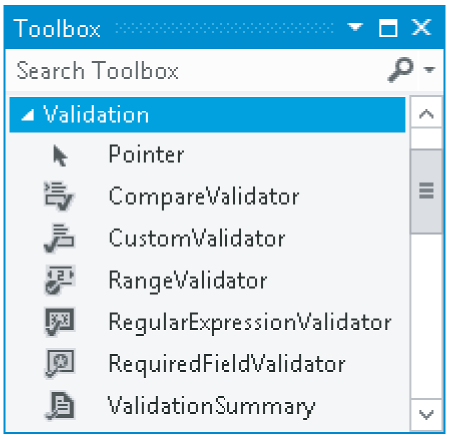
AdRotator
The *AdRotator* control enables you to display random advertisements on your website. The ads come from an XML file that you create on your server. Because it lacks advanced features like click tracking and logging that are required in most but the simplest scenarios, this control isn't used much in today's websites.

HiddenField
The *HiddenField* control enables you to store data in the page that is submitted with each request. This is useful if you want the page to remember specific data without the user seeing

it on the page. Because the field does show up in the HTML source of the page, and is thus accessible to the end user, you should never store any sensitive data in it.

HTML Controls
The HTML category of the Toolbox contains a number of HTML controls that look similar to the ones found in the Standard category. For example, you find the *Input* (Button) that looks like the *<asp:Button>*. Similarly, there is a Select control that has the *<asp:DropDownList>* and *<asp:ListBox>* as its counterparts. In contrast to the ASP.NET Server Controls, the HTML controls are client-side controls and end up directly in the final HTML in the browser. You can expose them to server-side code by adding a *runat="server"* attribute to them. This enables you to program against them from the Code Behind of a Web Form, to influence things like their visibility.
Data Controls Data controls were introduced in ASP.NET to offer an easy way to access various data sources like databases, XML files, and objects. Instead of writing lots of code to access the data source as you had to do in earlier versions of ASP.NET, you simply point your data control to an appropriate data source, and the ASP.NET run time takes care of most of the difficult issues for you.

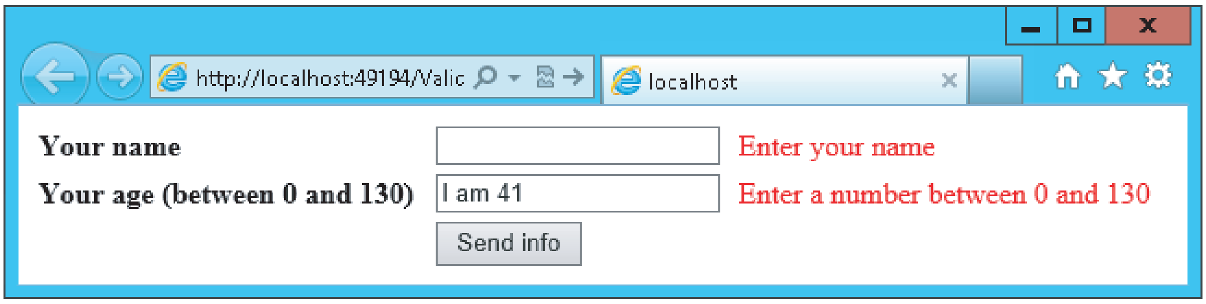
Validation Controls
Validation controls enable you to rapidly create Web Forms with validation rules that prohibit users from entering invalid data. For example, you can force users to enter values for required fields and check whether the entered data matches a specific format like a valid date or a number between 1 and 10. They even allow you to write custom code to create validation routines that are not covered by the standard controls. The beauty of the validation controls is that they can execute both on the client and the server, enabling you to create responsive and secure web applications.



Validating User Input in Web Forms
People concerned with validating user input. Although this may seem like paranoia at first, it is really important in any open system. Even if you think you know who your users are, and even if you trust them completely, they are often not the only users that can access your system. As soon as your site is out on the Internet, it’s a potential target for malicious users and hackers who will try to find a way into your system. In addition to these evil visitors, even your trustworthy users may send incorrect data to your server by accident.
To help you overcome this problem as much as possible, ASP.NET ships with a range of *validation controls* that help you validate data before it is used in your application.

The ASP.NET Validation Controls
ASP.NET 4.5 comes with six useful controls to perform validation in your website. Five of them are used to perform the actual validation, whereas the final control *ValidationSummary* is used to provide feedback to the user about any errors made in the page. Following figure shows the available controls in the Validation category of the Toolbox. The validation controls are extremely helpful in validating the data that a user enters in the system. They can easily be hooked to other controls like the *TextBox* or a *DropDownList*; however, they also support custom validation scenarios. Following figure demonstrates two of the validation

controls *RequiredFieldValidator* and *RangeValidator* at work to prevent a user from submitting the form without entering required and valid data.



Your name — Enter your name

Your age (between 0 and 130) — I am 41 — Enter a number between 0 and 130

The great thing about the validation controls is that they can check the input at the client and at the server. When you add a validation control to a web page, the control renders JavaScript that validates the associated control at the client. This client-side validation works on most modern web browsers with JavaScript enabled, including Internet Explorer, Firefox, Chrome, Opera, and Safari. At the same time, the validation is also carried out at the server automatically. This makes it easy to provide your user with immediate feedback about the data using client-side scripts, while your web pages are safe from bogus data at the server.

A Warning on Client-Side Validation

Although client-side validation may seem enough to prevent users from sending invalid data to your system, you should never rely on it as the only solution to validation. It's easy to disable JavaScript in the browser, rendering the client-side validation routines useless. In addition, a malicious user can easily bypass the entire page in the browser and send information directly to the server, which will happily accept and process it if you don't take countermeasures.

In general, you should see client-side validation as a courtesy to your users. It gives them immediate feedback so they know they forgot to enter a required field, or entered incorrect data without a full *postback* to the server. Server-side validation, on the other hand, is the only real means of validation. It's effectively the only way to prevent invalid data from entering your system. The following section discusses how you can employ the validation controls to protect your data.

Using the Validation Controls

To declare a validation control in your ASPX page, you use the familiar declarative syntax. For example, to create the *RequiredFieldValidator* control used in following figure, you need the following code:

```
<asp:RequiredFieldValidator ID="ReqVal1" runat="server"
ControlToValidate="YourName" ErrorMessage="Enter your name" />
```

The *ControlToValidate* property links this validation control to another control (YourName in this example) in the page. When asked to perform its validation, the validation control looks at the value of the linked control and when that value doesn't meet the validation rules you set, it displays the message set in the *ErrorMessage* property by default although you can override this behavior.

To give you an idea of how the validation controls work, the following example guides you through the process of using the *RequiredFieldValidator* in a contact form that is placed in a user control.

Using the RequiredFieldValidator

In this example create a user control called ContactForm.ascx. You can place it in a web page so visitors to your site can leave some feedback.

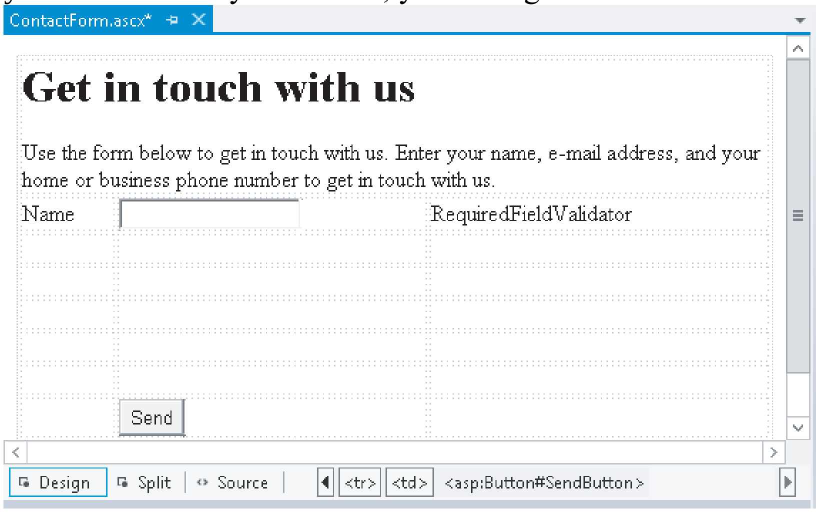
1. Open the New Web Application project and add a new user control in the application. **ContactForm.ascx**. Make sure that it uses your programming language and a Code Behind file.

2. Switch to Design View and insert a table by choosing Table Insert Table. Create a table with eight rows and three columns.

3. Merge the three cells of the first row. To do this, select all three cells, right-click the selection, and choose Modify Merge Cells.

4. In the merged cell, type some text that tells your users they can use the contact form to get in touch with you. You could use an h1 element as a heading above the page to draw the user's attention.

5. In the first cell of the second row, type the word **Name**. Into the second cell of the same row, drag a *TextBox* and set its *ID* to **Name.** Into the last cell of the same row, drag a *RequiredFieldValidator* from the Validation category of the Toolbox. Finally, into the second cell of the last row, drag a *Button*. Rename the button to **SendButton** by setting its ID and set its *Text* property to **Send**. When you're done, your Design View looks like following figure.



ContactForm.ascx*

Get in touch with us

Use the form below to get in touch with us. Enter your name, e-mail address, and your home or business phone number to get in touch with us.

Name RequiredFieldValidator

Send

Design | Split | Source | `<tr>` `<td>` `<asp:Button#SendButton>`

6. Click the *RequiredFieldValidator* once in Design View and then open up its Properties Grid by pressing F4. Set the following properties on the control:

PROPERTY	VALUE
CssClass	ErrorMessage
ErrorMessage	Enter your name
Text	*
ControlToValidate	Name

Note: you can type in the value for *ControlToValidate* directly or you can pick it from the list by clicking the down arrow.

7. Save the changes to the user control and then close it because you're done with it for now.

8. Open Contact.aspx from the About folder in Markup View and from the Solution Explorer, drag the user control ContactForm.ascx between the tags of the *cpMainContent* control. You should end up with this control declaration:

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" runat="Server">
<ContactForm ID="ContactForm" runat="server" />
</asp:Content>
```

Visual Studio remembers the last custom prefix you used and reuses that when dragging the user control onto the page.
9. Open *Web.config* and add or modify the following code under the <*appSettings*> element (which is a direct child of the main <*configuration*> node):

```
<configuration>
<appSettings>
<add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
</appSettings>
...
</configuration>
```

10. Save your changes and open Contact.aspx in your browser. If you get an error, make sure you renamed the *TextBox* to **Name** and that you set the *ControlToValidate* property on the *RequiredFieldValidator* to **Name.**
11. Leave the Name text box empty and click the Send button. Note that the page is not submitted to the server. Instead, you should see a red asterisk appear at the very right of the row for the name field to indicate an error. If the asterisk is not red, press Ctrl+F5 or Ctrl+R and click the Send button again.
12. Enter your name and click Send again. The page now successfully posts back to the server.

The Standard Validation Controls
The five validation controls (the ones in the Validation category of the Toolbox whose names end in Validator) ultimately all inherit from the same base class, and thus share some common behavior. Four of the five validation controls operate in the same way, and contain built-in behavior that enables you to validate associated controls. The last control, the *CustomValidator,* enables you to write custom validation rules not supported out of the box. The following table lists a number of common properties that are shared by the validation controls and that you typically use when working with them.

RangeValidator
The *RangeValidator* control enables you to check whether a value falls within a certain range. The control is able to check data types like strings, numbers, dates, and currencies. For example, you can use it to make sure a number is between 1 and 10, a character between A and F, or a selected date falls between today and the next two weeks. The following table lists its most important properties.
The following example shows a *RangeValidator* that ensures that the value entered in the Rate text box is a whole number that lies between 1 and 10:

```
<asp:RangeValidator ID="RangeValidator1" runat="server"
ControlToValidate="Rate" ErrorMessage="Enter a number between 1 and 10"
MaximumValue="10" MinimumValue="1" Type="Integer" />
```

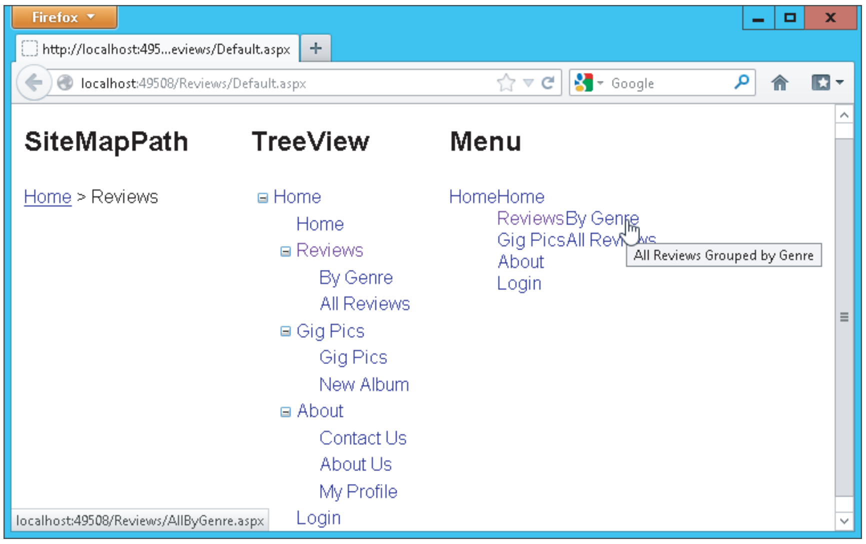
RegularExpressionValidator

The *RegularExpressionValidator* control enables you to check a value against a *regular expression* that you set in the *ValidationExpression* property of the control. Regular expressions offer a compact syntax that enables you to search for patterns in text strings. Regular expressions
are a complex subject, but fortunately, Visual Studio comes with a few built-in expressions that make it easy to validate values like e-mail addresses and ZIP codes.
The following example shows a *RegularExpressionValidator* control that ensures that a user enters a value that looks like an e-mail address:
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" ControlToValidate="Email" ErrorMessage="Enter a valid e-mail address" ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" />

CompareValidator
The *CompareValidator* can be used to compare the value of one control to another value. This is often used in sign-up forms where users have to enter a password twice to make sure they type the same password both times. Instead of comparing to another control, you can also compare against a constant value. This example shows a *CompareValidator* that ensures that two *TextBox* controls contain the same password:
<asp:CompareValidator ID="CompareValidator1" runat="server" ControlToCompare="ConfirmPassword" ControlToValidate="Password" ErrorMessage="Your passwords don't match" />
In the following example, you see most of these controls at work, except for the *RangeValidator.*
However, its usage is similar to the other validation controls, so it’s just as easy to add it to your web page or user control when you need it.

Navigation Controls
The controls you find under the Navigation category of the Toolbox are used to let users find their way through your site. The *TreeView* control presents a hierarchical display of data and can be used to show the structure of your site, giving easy access to all the pages in the site. The *Menu control* does a similar thing and provides options for horizontal and vertical fold-out menus. The *SiteMapPath* control creates a ―breadcrumb trail‖ in your web pages that enables your users to easily fi nd their way up in the hierarchy of pages in your site.

USING THE NAVIGATION CONTROLS
ASP.NET 4.5 offers three useful navigation tools: *SiteMapPath, TreeView,* and *Menu.* Following figure shows basic examples of the three navigation controls.
The *SiteMapPath* on the left shows the user the path to the current page. This helps if users want to go up one or more levels in the site hierarchy. It also helps them to understand where they are. The *TreeView* can display the structure of your site and enables you to expand and collapse the different nodes. The *Menu* control on the right initially only displays the Home menu item. However, as soon as you move the mouse over the menu item, a submenu appears.



Architecture of the Navigation Controls
To make it easy to show relevant pages in your site using a Menu, a TreeView, or a SiteMapPath, ASP.NET uses an XML-based file that describes the *logical structure* of your website. By default, this file is called Web.sitemap. This file is then used by the navigation controls in your site to present relevant links in an organized way. Simply by hooking up one of the navigation controls to the Web.sitemap file, you can create complex user interface elements like fold-out menus or a tree view.
Examining the Web.sitemap File
By default, you should call the site map file Web.sitemap. This enables the controls to find the right file automatically. For more advanced scenarios you can have multiple site map files with different names, with a configuration setting in the Web.config file that exposes these additional files to the system. In most cases, a single site map file is sufficient. A basic version of the site map file can look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
<siteMapNode url="~/" title="Home" description="Go to the homepage">
<siteMapNode url="~/Reviews" title="Reviews"
description="Reviews published on this site" />
<siteMapNode url="~/About" title="About"
description="About this site" />
</siteMapNode>
</siteMap>
```

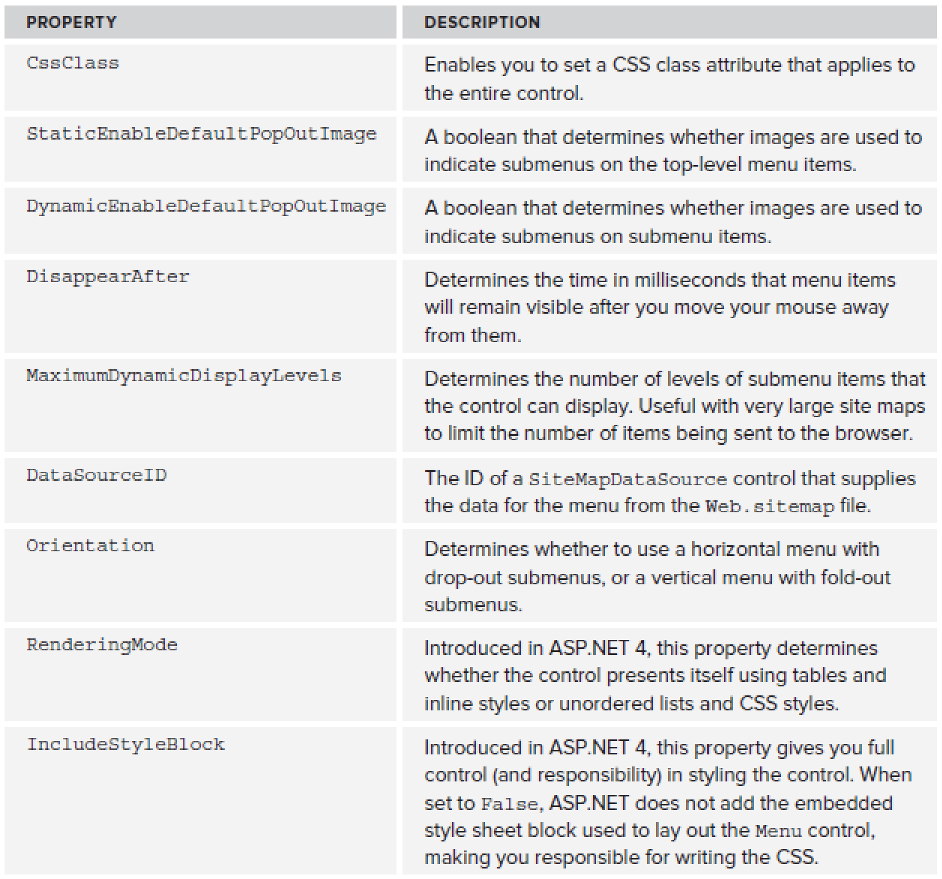
The site map file contains siteMapNode elements that together form the logical structure of your site. In this example, there is a single root node called Home, which in turn contains two child elements, Reviews and About.

Key Elements of the Web.sitemap File
Each siteMapNode can have many child nodes (but there can only be one siteMapNode directly under the siteMap element), enabling you to create a site structure that can be both wide and deep. The siteMapNode elements in this example have three of their attributes set: url, title, and description. The url attribute should point to a valid page in your website. You can use the syntax you saw in the previous section to refer to application-root–based URLs. The ASP.NET run time doesn't allow you to specify the same URL more than once, but you can work around that by making the URL unique by adding a query string. For example, ~/Login.aspx and ~/Login.aspx? type=Admin will be seen as two different pages.

The title attribute is used in the navigation controls to display the name of the page. The description attribute is used as a tooltip for the navigation elements. Figure shows a tooltip for the By Genre item. The navigation controls work together with the ASP.NET security mechanism. That is, you can automatically hide elements from controls like the Menu that users don't have access to.
The SiteMapPath control that displays a breadcrumb is able to find the Web.sitemap file itself. For the other two navigation controls, you need to specify a SiteMapDataSource control (which you can find under the Data category of the Toolbox) explicitly as an intermediate layer to the Web.sitemap file. To create a Web.sitemap file, you need to add one to your site and then manually add the necessary siteMapNode elements to it. There is no automated way in Visual Studio to create a site map file based on the current site's structure, although third-party solutions exist that help you with this.

Using the Menu Control

The Menu control is very easy to use and tweak. To create a basic menu, all you need to do is add one to your page, hook it up to a SiteMapDataSource control, and you're done. But at the same time, the control is quite flexible and has around 80 public properties (including the ones shared by all controls) that enable you to tweak every visual aspect of the control. The following table lists the most common properties used with the menu.



PROPERTY	DESCRIPTION
`CssClass`	Enables you to set a CSS class attribute that applies to the entire control.
`StaticEnableDefaultPopOutImage`	A boolean that determines whether images are used to indicate submenus on the top-level menu items.
`DynamicEnableDefaultPopOutImage`	A boolean that determines whether images are used to indicate submenus on submenu items.
`DisappearAfter`	Determines the time in milliseconds that menu items will remain visible after you move your mouse away from them.
`MaximumDynamicDisplayLevels`	Determines the number of levels of submenu items that the control can display. Useful with very large site maps to limit the number of items being sent to the browser.
`DataSourceID`	The ID of a `SiteMapDataSource` control that supplies the data for the menu from the `Web.sitemap` file.
`Orientation`	Determines whether to use a horizontal menu with drop-out submenus, or a vertical menu with fold-out submenus.
`RenderingMode`	Introduced in ASP.NET 4, this property determines whether the control presents itself using tables and inline styles or unordered lists and CSS styles.
`IncludeStyleBlock`	Introduced in ASP.NET 4, this property gives you full control (and responsibility) in styling the control. When set to `False`, ASP.NET does not add the embedded style sheet block used to lay out the `Menu` control, making you responsible for writing the CSS.

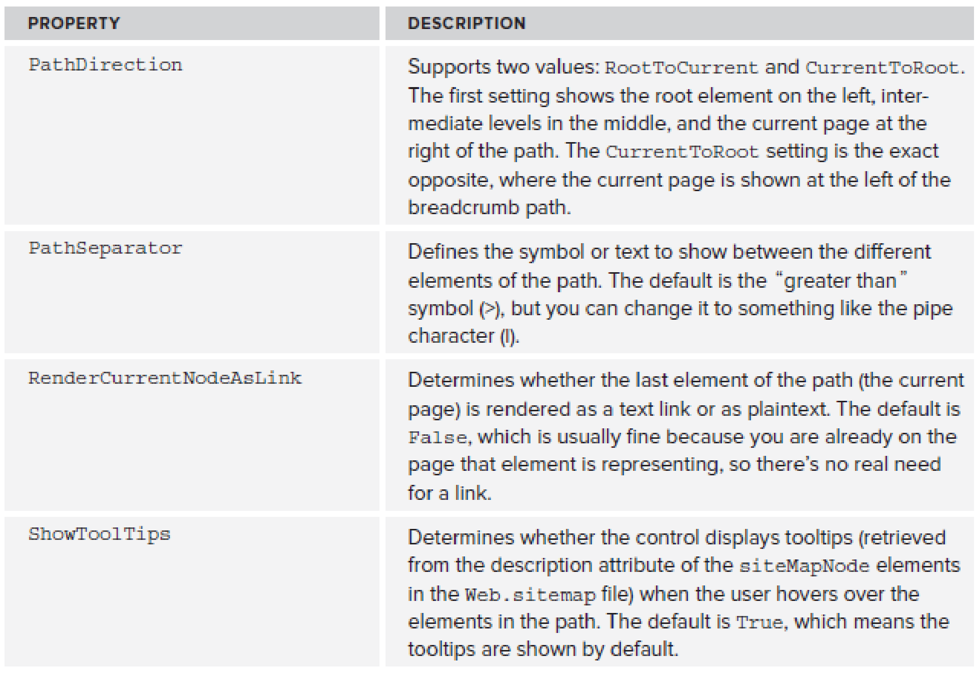
The Menu control also has a few properties that start with Static or Dynamic, two of which were shown in the preceding table. The Static properties are used to control the main menu items that appear when the page loads. Because they don't change or get hidden when you hover over them, they are considered static. The submenus are dynamic, because they appear only when you activate the relevant main menu items.
In addition to these properties, the Menu control also has a number of style properties that enable you to change the look and feel of the different parts of the menu.

Using the TreeView Control

A TreeView is capable of displaying a hierarchical list of items, similar to how the tree in Windows Explorer looks. Items can be expanded and collapsed with the small plus and minus icons in front of items that contain child elements. This makes the TreeView an ideal tool to display the site map of the website as a means to navigate the site. The data used by the TreeView control is not limited to the Web.sitemap file, however. You can also bind it to regular XML files and even create a TreeView or its items (called nodes) programmatically. The following table lists the most common properties of the TreeView. Again, the MSDN online
help is a good place to get a detailed overview of all the available properties and their descriptions. The TreeView control has a number of style properties that enable you to change the look and feel of the different parts of the tree. To tell the TreeView which items to show, you bind it to a SiteMapDataSource control, which is demonstrated next.

Using the SiteMapPath Control

The SiteMapPath control shows you where you are in the site's structure. It presents itself as a series of links, often referred to as a *breadcrumb*. It's a pretty simple yet powerful control with more than 50 public properties you can set through the Properties Grid to influence the way it looks. Just like the Menu and TreeView, it has a number of style properties you use to change the look of elements like the current node, a normal node, and the path separator.
The following table lists a few of the most common properties of the SiteMapPath control.



PROPERTY	DESCRIPTION
`PathDirection`	Supports two values: `RootToCurrent` and `CurrentToRoot`. The first setting shows the root element on the left, intermediate levels in the middle, and the current page at the right of the path. The `CurrentToRoot` setting is the exact opposite, where the current page is shown at the left of the breadcrumb path.
`PathSeparator`	Defines the symbol or text to show between the different elements of the path. The default is the "greater than" symbol (>), but you can change it to something like the pipe character (|).
`RenderCurrentNodeAsLink`	Determines whether the last element of the path (the current page) is rendered as a text link or as plaintext. The default is `False`, which is usually fine because you are already on the page that element is representing, so there's no real need for a link.
`ShowToolTips`	Determines whether the control displays tooltips (retrieved from the description attribute of the `siteMapNode` elements in the `Web.sitemap` file) when the user hovers over the elements in the path. The default is `True`, which means the tooltips are shown by default.

Depending on your personal preferences, you usually don't need to defi ne any of the styles of the SiteMapPath control. In the final page in the browser, the SiteMapPath consists of mainly anchor tags (<a>) and plaintext. If you have set up a specifi c selector for anchors in your CSS file, the SiteMapPath automatically shows itself in line with the other links in the page.