From HTML to ASP.NET

References : MSDN , ASP.Net For Beginners , ADO.Net Using C# , Thinking in C++ , ASP.Net Programming 2008

Your browser’s main job is requesting pages from the server and displaying what the server has processed. Using HTTP as a transfer protocol, your browser parses (interprets) the HTML code it gets from the server, but otherwise does little processing. Because your HTTP client (browser) leaves most of the processing work to the server, it is considered a *thin client.* There’s nothing wrong with a thin client, and it can process JavaScript. With plug-ins, which virtually all browsers have built in, a thin client can also process certain kinds of files such as SWF (compiled Adobe Flash files), Java Applets, ActiveX Controls, as well as other files requiring that the browser have compatible plug-ins



With a server-side program to *catch* the data that is sent to the server, lots of processes on the server are possible. Suppose the user enters the name, *Willie B. Goode.* The server-side file can use the property name, username, to extract a value. In this case, it would extract the name Willie B. Goode and do something with it. It might run a SQL script to store the name in database, compare it with a password, or it could pass information about Willie B. Goode back to the browser in HTML



.Net Organization

Probably the best way to think of .NET is as a big control system with two key elements. First is the .NET framework class library. This library has thousands of classes organized into *namespaces.* This large set of classes was designed to provide all the class support for virtually any application a developer would want to create. This library is a repository of reusable types for object-oriented development. For example, System.Web.UI Namespace allows you to create several different ASP.NET server controls and user interfaces. A *namespace* can be understood as an organizing concept for classes. Each namespace contains a collection of classes, most of which you do not need for any single application The following represent a *few* of the classes in this namespace:

● Control

● Data Binder

● PageParser

● UserControl

To use the classes as well as other structures such as interfaces, you need to include the namespace in your program.

Second, and at the core of .NET, is the *common language runtime.* The runtime is a management agent for code as it executes. All of the services, such as memory and thread management and remoting, are handled by the .NET framework. For example, VB.NET (Visual Basic) has access to the same family of classes as does C#. It is designed to aid in strict type safety and other structures that secure both robustness and security. Microsoft refers to code that targets the common language runtime as *managed code* and refers to code that does not target the runtime as *unmanaged code*



**Q: Why are ASP.NET and C# developed in separate windows if they are part of the**

**same program? Aren’t most programs developed in a single file?**

**A:** ASP.NET is a set of forms and controls, while C# is a dynamic language that communicates not only with ASP.NET but also with databases and outside links. Think of the separation as you would an automobile. The car without the engine focuses on the arrangement of seats and the different controls that make a car ready to be used for transportation. That is the ASP.NET aspect of the code behind. The car’s engine is what makes it go. It is a different element than the car’s body and takes a different kind of thinking to work on it. Nevertheless, the engine interacts with the car body and responds to its controls. C# is like the engine, and by keeping it separate during development, you can clearly focus on what you want it to do.

ASPX and C# Files

First, look at the ASP.NET as a whole.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"

Inherits="\_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

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

<head runat="server">

<title>Untitled Page</title>

</head>

<body>

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

the first line has far more information than a typical HTML one. It has been broken down into three segments:

<%@ Page Language="C#"

This first segment lets the page know that the language in use is C# and not VB or some

other language. It’s pretty self-explanatory. The next segment has a special Boolean, AutoEventWireup, and requires a bit more

explanation:

AutoEventWireup="true"

The AutoEventWireup Boolean indicates whether the ASP.NET pages are automatically connected to event-handling functions. With C#, the default is set to true,

so the pages are connected to event-handling functions

Visual Studio 2008 automatically assigns the ASP.NET filename to the code file to use, and since the name of the ASP.NET file is Default.aspx the CodeFile is Default.aspx.cs. That part is fairly straightforward.

CodeFile="Default.aspx.cs" Inherits="\_Default" %>

The Inherits attribute refers to the name of the class that the code inherits. This may

be a little confusing because in C# the reference for one class inheriting another is simply

a colon (:). For example, in the C# code you will see the line

public partial class \_Default : System.Web.UI.Page

using System;

using System.Data;

using System.Configuration;

using System.Linq;

using System.Web;

using System.Web.Security;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.UI.WebControls.WebParts;

using System.Web.UI.HtmlControls;

using System.Xml.Linq;

public partial class \_Default : System.Web.UI.Page

{

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

{

Label1.Text = "This ASP.NET 3.5 application written by Your Name";

}

}

The C# class definition begins with the line

public partial class \_Default : System.Web.UI.Page

This statement announces that the *name* of the class is \_Default and that it inheritsSystem.Web.UI.Page. From a strictly C# point of view, the partial class allows you tobreak up a class into more than a single file. This is useful where you have a big class or you have multiple developers working on a single class with different assignments. When working with ASP.NET (ASPX) files and C#, you can think of the ASPX file and CS files as the source code for multiple parts of the partial class. Strictly speaking that’s not wholly accurate, but for all practical purposes it’s a useful way of thinking about the value of a partial class in C# with an ASP.NET file. More accurately, the partial class allows C# to declare all the server-side controls you have declared in your ASPX file. Also, the partial class allows automatically generated code to be stored in a temp file from the code you generate.

Passing Values Between ASP.NET and C#

Using the partial class format that connects ASP.NET to C#, you must be aware thatnames created in ASP.NET are used by C#. Therefore, in any discussion of C# andASP.NET, you have to understand how the naming works Take, for example, the following task:

*User inputs two values. The first value is multiplied by .85 and the second by .15.The output shows the sum of the products.*

P.H.W : Write Web Page to simulate this Example

***Variable* :** a *variable* is a container-like structure used to store data. The data in the container can change in value—it varies; hence, the term *variable*

***Constant*** : is like a variable in that it stores values, but unlike a variable, a constantrepresents an unchanging value. For example, the value of the mathematical constant *pi* is unchanging

Example :

const int VAGE = 18;

int yourAge = 35;

int legalAge;

legalAge = yourAge - VAGE; //Expression using both constant and variable

***Arrays*** : are data structures that contain multiple elements. Each element of an array is

itself an object or variable with values. Each of the values in each of an array’s elements must be of the same kind. For example, an integer type array can only contain integers, and a string type array can only contain strings

type[] arrayLabel;

For instance, the following creates a string array named stringSlots:

string[] stringSlots;

Or Even

string[] stringSlots = new string[3] { "apples" , "oranges" , "pears" };

string[] stringSlots = { "apples" , "oranges" , "pears" };

Creating and Using Classes

you can build your own classes in C#. In C#, as in all object-oriented programming (OOP) languages, a class is a way of abstracting something. For example, you can abstract an animal like a cat. All cats have certain characteristics that are similar, but the actual cats have unique values for those general characteristics. Actual cats in our example would be referred to as *instances* of the Cat object. For example, the same cat abstraction applies to both housecats who rely on their owners to feed and house them and to cheetahs fending for themselves in the wild. Each time you need a cat, you can use the cat class to provide the general properties, and then all you have to do is to specify the characteristics of those properties

Generally, the partial classes you’ve seen simply have a single event that defaults to Page\_Load. The rest of the class usually involves nothing more than placing some statements into the event handling function.A basic class typically has the following:

● List of **using** statements

● The class *name*

● The class *constructor*

● Variables (private instance variables)

● Properties (public accessors for the variables)

● Methods

The classes you will create are *public* ones. That means they can be accessed from other classes, including partial classes such as you’ve seen up to now. The following shows the generic features of a simple class (the key elements have been placed in boldface):

**using** System;

public class **MyClass**

{

private string **myProperty;**

public **MyClass()**

{

myProperty = "Some value";

}

public string **MyMethod**(string paramName)

{

return myProperty + paramName

}

}

Example :

OutClass.cs

using System;

/// Class with single method to send string

//Declaring a class with the class label

public class OutClass

{

//Property of the class

private string header;

//Constructor function has the same name as the class

public OutClass()

{

header = "From your class:";

}

//Class method

public string sendMsg(string fromClass)

{

return header + fromClass;

}

}