Creating ASPX Pages

Once your Web site has been created, the next step is to begin adding pages to the site. An ASP.NET page is also known as a Web Form and can be composed of a single file or a pair of files. The steps for adding a new Web page to a Web site are as follows:

1. Using the Visual Studio 2008 menus, request a new Web Form. Typically this is done inside Solution Explorer by right-clicking the Web site and selecting Add New Item.

2. In the Add New Item dialog box (see Figure 1-11), assign a name to the Web Form.

3. Select the programming language for this Web Form.

4. Indicate if the page is self-contained or uses a pair of files using the Place Code In

Separate File check box.

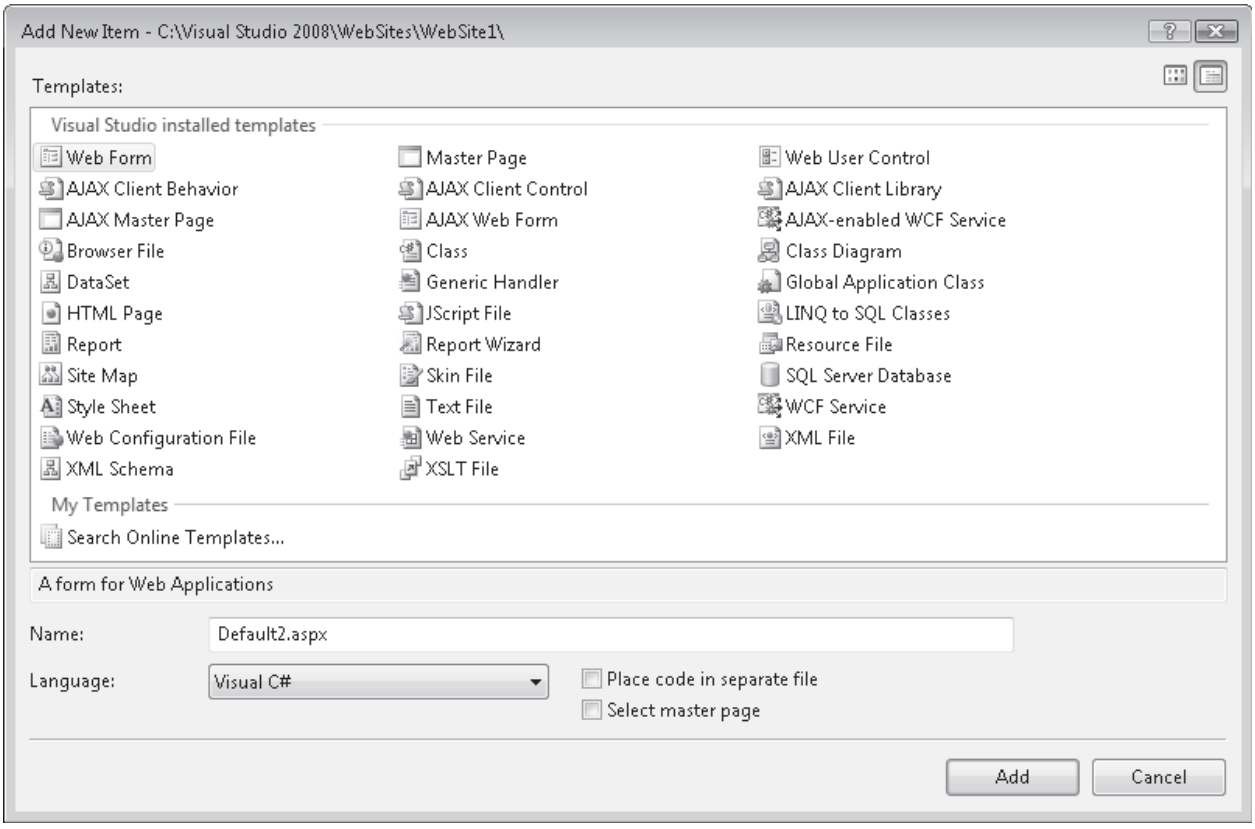


Figure 1-11 Adding a new Web Form to your Web site

The Place Code In Separate File check box allows you to indicate whether your page should be made up of a single, self-contained file or a pair of files (HTML layout and an associated code-behind file). Note that you can also select a master page on which to base the look of your page. Master pages allow you to create a consistent look and feel for your entire Web site.

The Anatomy of an ASPX Page

A page in ASP.NET contains user interface layout information, code that executes on the server, and directives to both connect the layout with the code and to tell ASP.NET how the page should be processed. The standard ASP.NET page has an .aspx extension.

The typical .aspx page includes three sections: page directives, code, and page layout. These sections are

defined as follows:

Page directives This section is used to set up the environment, specifying how the page should be processed. For example, this is where you can indicate an associated code file, development language, transaction, and more.

Code This section contains code to handle events that execute on the server based on the ASP.NET page processing model. By default, Visual Studio creates a separate file that contains your code. This is called a code-behind file and is attached to the .aspx page.

This file includes the .cs or .vb extension to indicate the file represents code (for example, Default.aspx.cs).

Code can be placed within the .aspx page itself (typically at the top of the file). To do so, you define *<script>* tags and place your code within them. The script tag should include the *runat="server"* attribute to denote server-side script. In this way you can create single files that contain your entire page.

Page layout The page layout is written using HTML. This includes the HTML body, markup, and style information. The HTML body may contain HTML tags, Visual Studio controls, user controls, and simple text.

The code for a simple, single-file Web page might look like Listing 1-1.

Listing 1-1: A Single-Page Web Form

```
<!--page directives-->

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

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

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!--code-->
<script runat="server">
private void OnSubmit(Object sender, EventArgs args) {
LabelReponse.Text = "Hello " + TextBoxName.Text;
}
</script>
<!--page layout-->
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Sample Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
Enter Name: <asp:TextBox ID="TextBoxName"
runat="server"></asp:TextBox>
<asp:Button ID="ButtonSubmit" runat="server" Text="Submit"
OnClick="OnSubmit" />
<br />
<asp:Label ID="LabelReponse"
runat="server" Text=""></asp:Label>
</div>
</form>
</html>
```

Notice the *runat=”server”* attribute is defined in the script block. This indicates that the code contained within the script block will run on the server (and not on the

client). On execution, ASP.NET will create server-side objects that contain this code as well as an instance of the *Page* class to contain the controls defined inside the page as instances of their given type (*System.Web.UI.WebControls.Textbox*, for example).

This server-side object will be invoked on user request and will execute code in response to events.

Single-File Versus Code-Behind Pages In the previous example, the Web page contains both server-side code and markup in a single file.

This is referred to as a single-file page in ASP.NET. You can create these types of pages by leaving the Place Code In Separate File check box cleared in the Add New Item dialog box.

Some developers are more comfortable working with single-file pages than the code-behind model.

In the single-file model, the compiler generates a new class for your page. This class inherits from the base *Page* class. It is typically named with the format ASP.pagename_aspx. This class contains control declarations, event handlers, and related code you have written for your page.

The code-behind programming model physically separates your user interface layout markup and your server-side code into two distinct files. In this case the .aspx page contains your layout markup and the related .aspx.cs or .aspx.vb file contains the associated code.

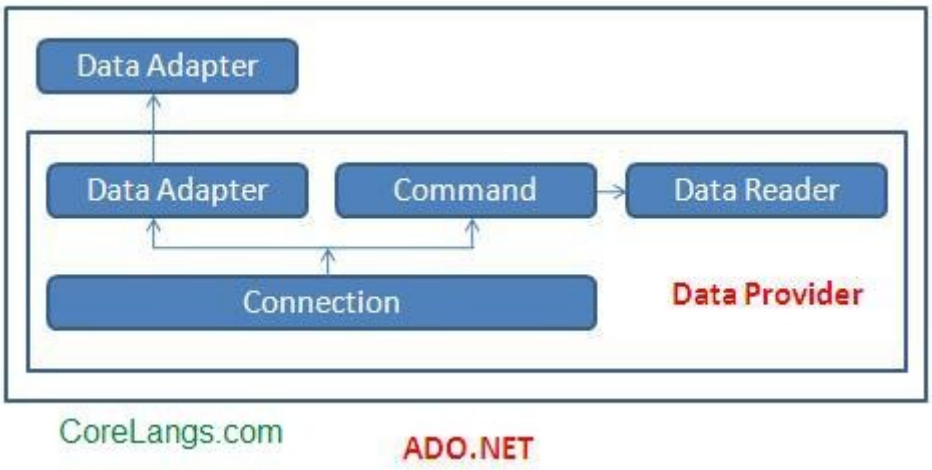
Developers who prefer to separate code from form layout will find this model easier to work with.

In the case of code-behind files, your code is stored in a partial class that inherits from the base *Page* class. Partial classes allow code-behind files to be dynamically compiled with their associated .aspx pages into a single class type. This means you no longer need to declare member variables in the code-behind page for each control, nor do you need this code getting in the way of your code. Instead, Visual Studio keeps control declaration and page initialization information in a separate partial class. This greatly simplifies maintenance of sites that are based on the code-behind model.

When your page gets compiled, the partial classes are merged to form yet another partial class. This class is then used by ASP.NET to generate another class (which inherits from the merged partial class) that is used by ASP.NET to build your Web page. Both these classes are then compiled into a single assembly similar to that of the single-file model.

ADO.NET Architecture

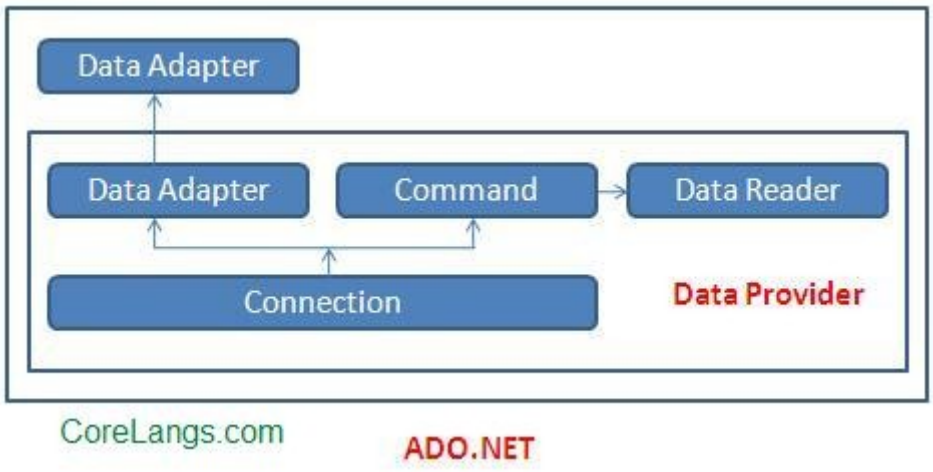
ADO.NET



ADO.NET consist of a set of Objects that expose data access services to the .NET environment. It is a data access technology from Microsoft .Net Framework , which provides communication between relational and non relational systems through a common set of components .

System.Data namespace is the core of ADO.NET and it contains classes used by all data providers. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code.

Data Providers and DataSet

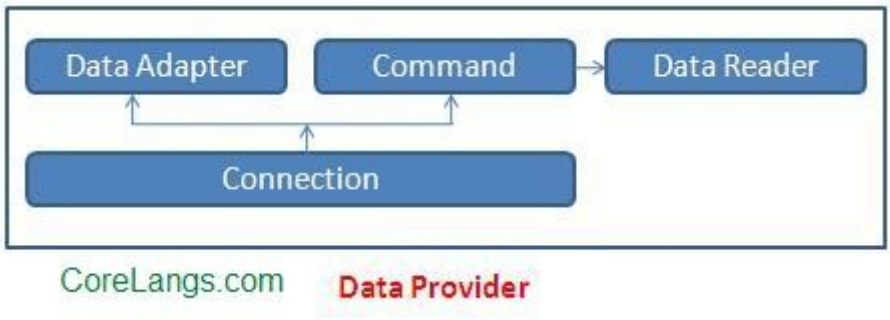


The two key components of ADO.NET are Data Providers and DataSet . The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases. DataSet class provides mechanisms for managing data when it is disconnected from the data source.

Data Providers

The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft SQL Server Data Provider , OLEDB Data Provider and ODBC Data Provider . SQL Server uses the SqlConnection object , OLEDB uses the OleDbConnection Object and ODBC uses

OdbcConnection Object respectively.



A data provider contains Connection, Command, DataAdapter, and DataReader objects. These four objects provides the functionality of Data Providers in the ADO.NET.

Connection

The Connection Object provides physical connection to the Data Source. Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.

Command

The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source. The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

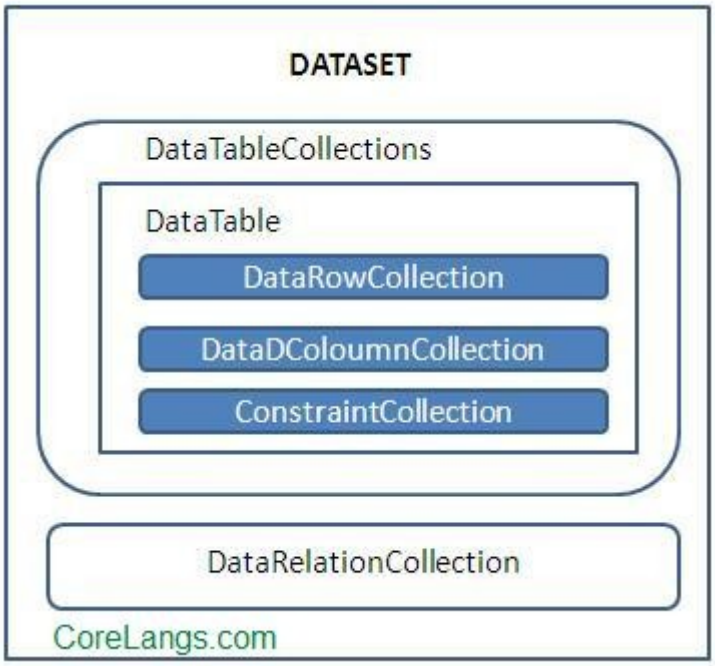
DataReader

The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data. DataReader requires a live connection with the databse and provides a very intelligent way of consuming all or part of the result set.

DataAdapter

DataAdapter Object populate a Dataset Object with results from a Data Source . It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.

DataSet



DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

DataSet contains rows, columns,primary keys, constraints, and relations with other DataTable objects. It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. The DataAdapter Object provides a bridge between the DataSet and the Data Source.

Object oriented concept in C# .net

Ans: Object oriented programming (OOP) is a programming structure where programs are organized around objects as opposed to action and logic. This is essentially a design philosophy that uses a different set of programming languages such as C#. Understanding OOP concepts can help make decisions about how you should design an application and what language to use.

Everything in OOP is placed together as self-sustainable "objects." An object is a combination of variables, functions, and data that performs a set of related activities. When the object performs those activities, it defines the object's behavior. In addition, an object is an instance of a class. Furthermore, C# offers full support for OOP including inheritance, encapsulation, abstraction, and polymorphism:

- **Encapsulation** is when a group of related methods, properties, and other members are treated as a single object.
- **Inheritance** is the ability to receive ("inherit") methods and properties from an existing class.
- **Polymorphism** is when each class implements the same methods in varying ways, but you can still have several classes that can be utilized interchangeably.
- **Abstraction** is the process by which a developer hides everything other than the relevant data about an object in order to simplify and increase efficiency.

We'll discuss each of these concepts in more detail in this post.

What is an Object?

Objects are instances of classes. In other words, an instance of a class is an object defined by that particular class. Creating a new instance, or an object, is called instantiation. This is how you define a class:

```
C#

class SampleClass

{

}
```

A light version of classes in C# is called structures. These are beneficial when you want to create a large array of objects but don't want to overwhelm your available memory. A class is made up of three things:

- A name
- Operations
- Attributes

Objects are created from a class blueprint, which defines the data and behavior of all instances of that type.

```
public class Student

{

}
```

Here's another example:

```
Class Example

{

/* fields,

Variables,

Methods,

Properties,

*/

}
```

Here's an example of an object:

```
Example exmplObject = new Example();
```

In memory, you can create an object using the "new" keyword. In C#, value types refer to other data type variables while objects are reference types. Moreover, other value types are stored in the stack while objects are stored in the heap.

Keep in mind that everything in C# is a class. An object is a section of memory that has been configured according to the class blueprint. Every instance, or object, of a particular class has access to several methods and properties of that class.

Data Abstraction

This provides essential features without describing any background details. Abstraction is important because it can hide unnecessary details from reference objects to names. It is also necessary for the construction of programs. Instead of showing how an object is represented or how it works, it focuses on what an object does. Therefore, data abstraction is often used for managing large and complex programs.

Encapsulation

This binds the member function and data member into a single class. This also allows for abstraction. Within OOP, encapsulation can be achieved through creating classes. Those classes then display public methods and properties. The name encapsulation comes from the fact that this class encapsulates the set of methods, properties, and attributes of its functionalities to other classes.

Polymorphism

This is the ability of an object to perform in a wide variety of ways. There are two types:

1. Dynamic polymorphism (runtime time). You can obtain this type through executing function overriding.

2. Static polymorphism (compile time). You can achieve static polymorphism through function overloading and operator overloading.

Within OOP, polymorphism can be achieved using many techniques including:

- Method overloading (defining several methods at the same time)
- Method overriding (this allows a subclass to override a specific implementation of a method already issued by one of its super-classes)
- Operator overloading (some or all of the operators are handled has polymorphic functions with different behaviors depending on the types of its arguments)

Inheritance

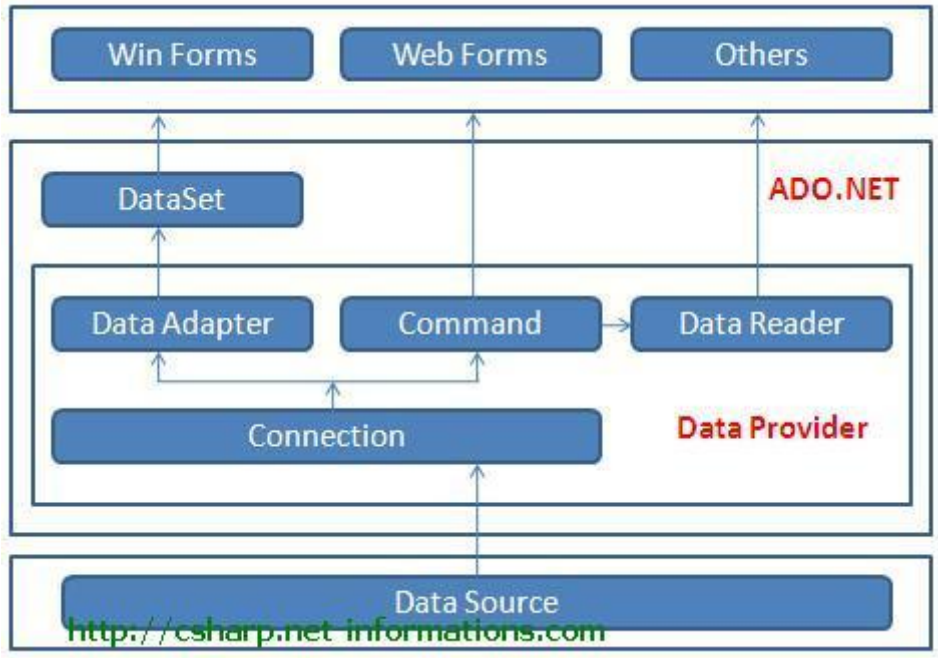
Through inheritance, a class can "inherit" the characteristics of another general class. To illustrate, dogs are believed to be the descendants of wolves. All dogs have four paws and can hunt their prey. This function can be coded into the Wolf class. All of its descendants can use it. Inheritance is also an is-kind-of relationship. For instance, a golden retriever is a kind of animal. Here's an example:

```
class BaseClass



{



}



class DerivedClass : BaseClass
```

```
{



}
```

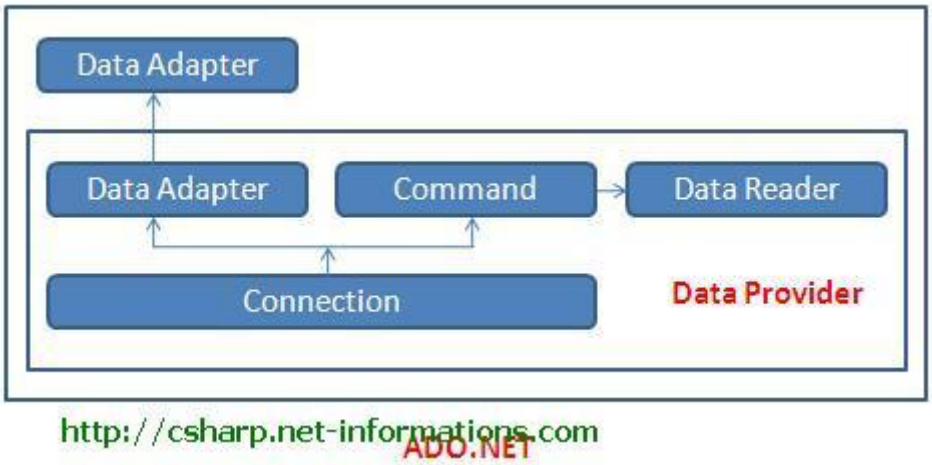
Architecture of ADO.net and Data set

ADO.NET



ADO.NET is a data access technology from Microsoft .Net Framework , which provides communication between relational and non-relational systems through a common set of components . **ADO.NET** consist of a set of Objects that expose data access services to the .NET environment. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate**ADO.NET** data access code.

Data Providers and DataSet



The two key components of ADO.NET are **Data Providers** and **DataSet** . The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft **SQL Server Data Provider** , **OLEDB Data Provider** and **ODBC Data Provider** . SQL Server uses the SqlConnection object , OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.

C# SQL Server Connection

C# OLEDB Connection

C# ODBC Connection

The four Objects from the .Net Framework provides the functionality of Data Providers in the ADO.NET. They are **Connection** Object, **Command** Object ,**DataReader** Object and **DataAdapter** Object. The Connection Object provides physical connection to the Data Source. The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source. The DataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data. Finally the DataAdapter Object , which populate a Dataset Object with results from a Data Source .

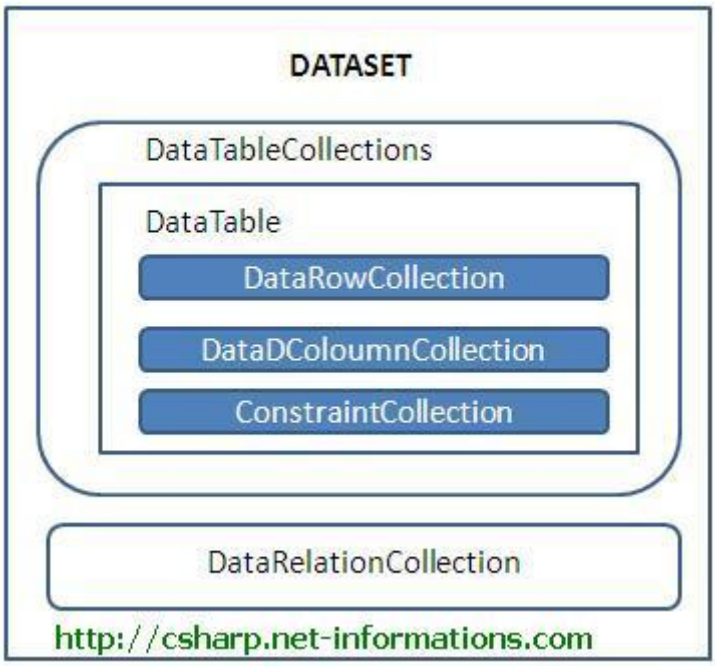
C# Connection

C# Command

C# DataReader

C# DataAdapter

DataSet



DataSet provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

DataSet consists of a collection of **DataTable** objects that you can relate to each other with DataRelation objects. The DataTable contains a collection of **DataRow** and**DataCoulumn** Object which contains Data. The DataAdapter Object provides a bridge between the DataSet and the Data Source.

Design and application in C#.net which is console based

Console programs are easy to develop, because console based applications perform all their input and output at the command line, they are ideal for quickly trying out language features and writing command-line utilities. If you have ever learned a programming language, you know that they all start with the "Hello, world!" program. Here also we start with the "Hello, world!".

Hello World - Your First Program

In order to create a C# console based application

1. Open your Visual Studio

2. On the File menu, click New Project.

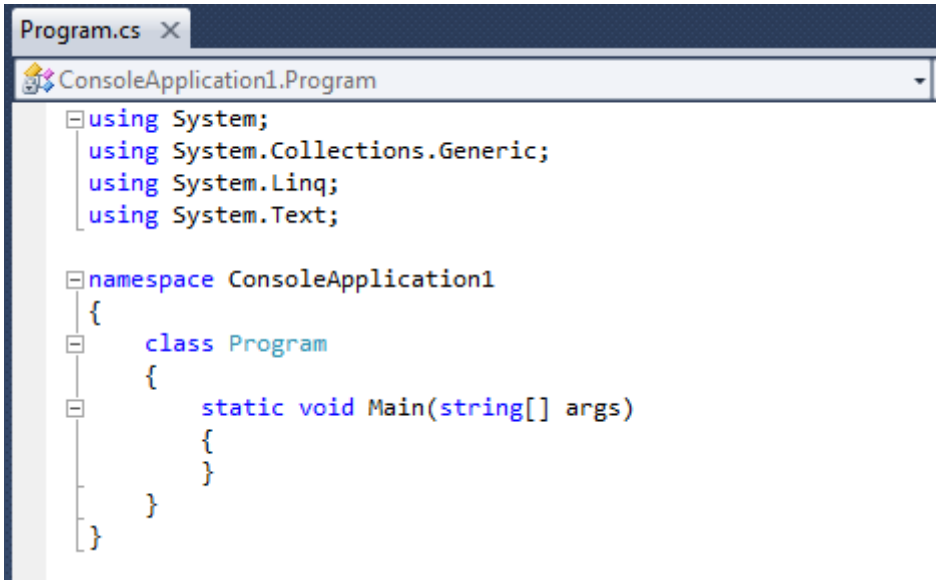
Then the New Project dialog box appears. This dialog box lists the different default application types.

3. Select Console Application as your project type and change the name of your application at the bottom textbox.

If you are npt comfortable with default location, you can always enter a new path if you want.

4. Then Click OK.

After click OK button, you will get a screen like the following picture.



Here in the following program, we just print a "Hello, world!" message only. So copy and paste the following command in the main code block.

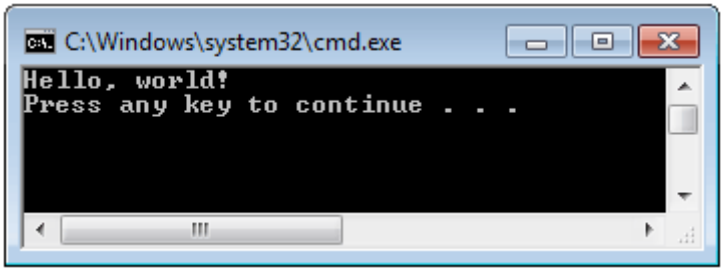
```
Console.WriteLine("Hello, world!");
```

Here you can see the full source code.

```
using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApplication1

{
```

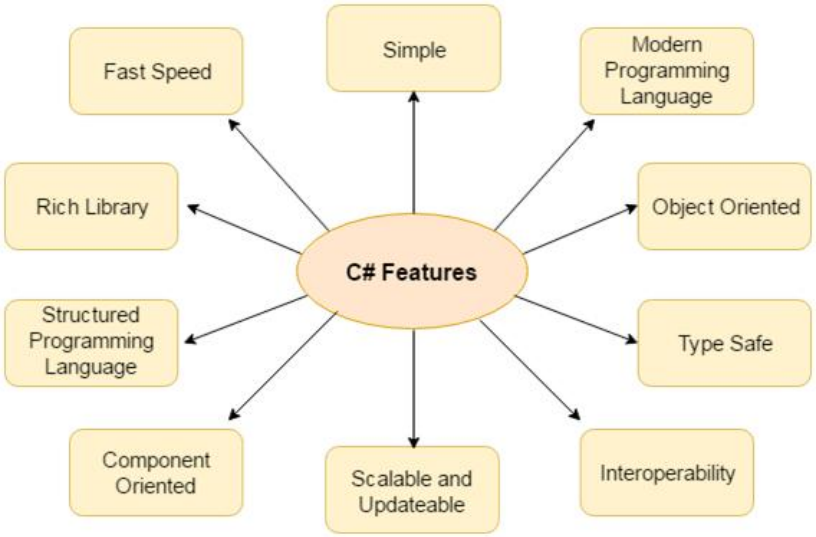
```
class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello, world!");

}

}

}
```

After enter the command, the next step is to run the program. You can run your program using Ctrl+F5 . Then Visual Studio will keep the console window open, until you press a key. You will get screen look like the following picture.



Now you created your first program in Visual Studio.

Features of C#.net



1) Simple

C# is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2) Modern Programming Language

C# programming is based upon the current trend and it is very powerful and simple for building scalable, interoperable and robust applications.

3) Object Oriented

C# is object oriented programming language. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grow.

4) Type Safe

C# type safe code can only access the memory location that it has permission to execute. Therefore it improves a security of the program.

5) Interoperability

Interoperability process enables the C# programs to do almost anything that a native C++ application can do.

6) Scalable and Updateable

C# is automatic scalable and updateable programming language. For updating our application we delete the old files and update them with new ones.

7) Component Oriented

C# is component oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.

8) Structured Programming Language

C# is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

9) Rich Library

C# provides a lot of inbuilt functions that makes the development fast.

10) Fast Speed

The compilation and execution time of C# language is fast.

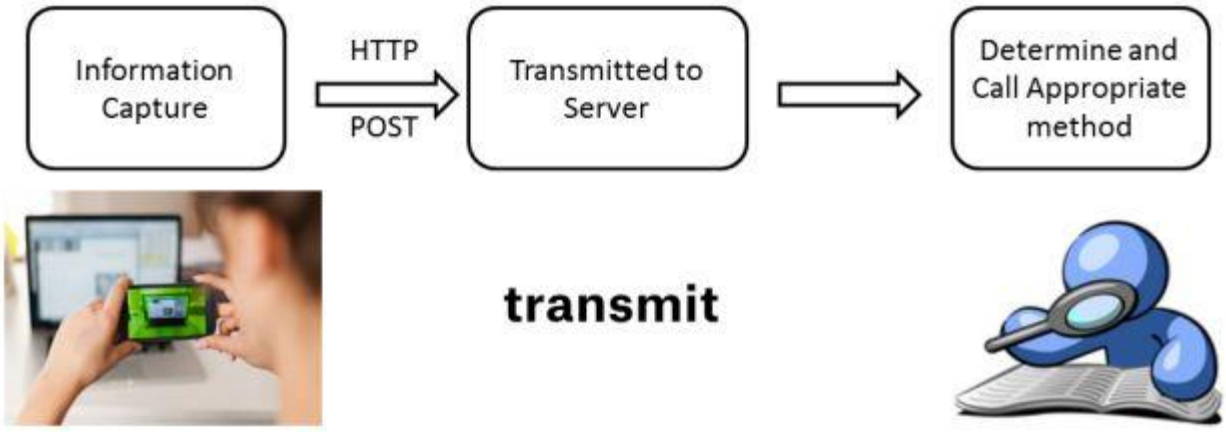
Event Handling In ASP.NET

In ASP.NET Web forms, we are using an event-based model. We can create every server control event. For example, in the case of the button control, we can create an event handler to handle the click event. Even the ASPX page has its page life-cycle events. We all know that when an ASPX page is requested by the user, all its controls render as HTML controls. In this session, we are going to learn that there is a mechanism at work for the event handling.

Event is a way for a class to provide the notifications to the clients of the class when something interesting happens to an object. We are basically familiar with the use of events in a user interface (UI) such as button click, changed text, etc. An event is a useful way for the objects to signal the state changes that may be useful to the clients of the object.

In simple terms, it's like giving the instruction for an action.

Therefore, let's see how our familiar action, such as button click, works for us. Every event associated with the server controls originate on the client browser but are handled on the Web Server by an ASP.NET page. How does ASP.NET handle this?



Events we come across in our coding:

- Events for the Server Controls.
    1. Protected **void** ButtonID_Click(Object sender,EventArgs e)
- Events for the Page Life Cycle.
    1. **Protected void** Page_Load(Object sender,EventArgs e)
- Application/Session Life Cycle Events.
    1. **Protected void** Application_Start(**object** sender, EventArgs e)

Each function has three things in common:

1. Object Argument - It represents the object that raised the event.
2. Event Args - Event object contains any event specific information.
3. Name of Event - All these follow a naming convention.

Hence, start with the naming convention, it helps ASP.NET to handle the event.

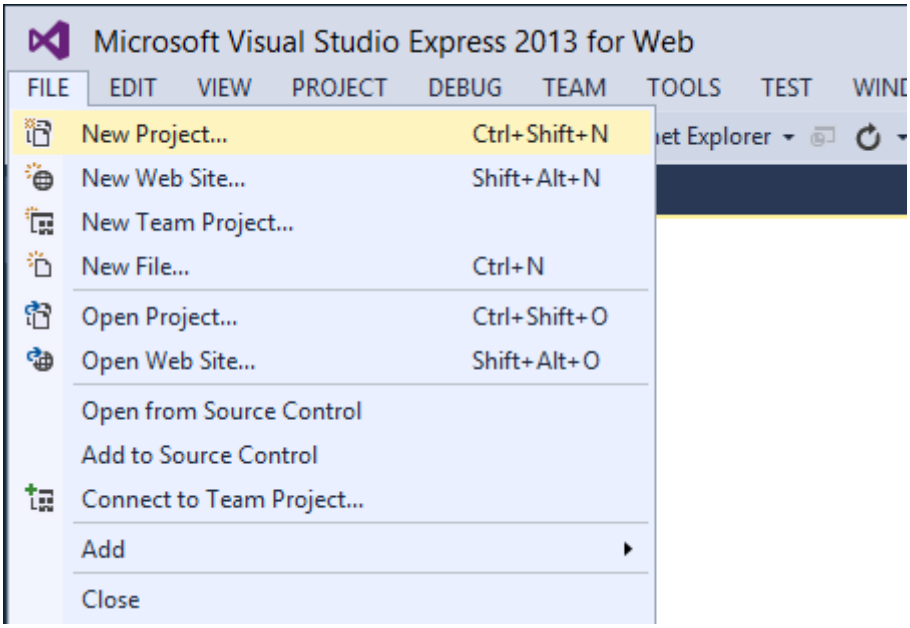
There is a property on the page level directive called AutoEventWireUp, which controls the automatic binding of the page events, based on the method of naming convention.

Another thing that works for us is Postback; the process of submitting the ASP.NET page back to the server (simply a subsequent request to a page). It says to the server that the client is posting data back to you for the processing. Some controls have AutoPostBack properties, while some do not. In this case, we define AutoPostBack property externally as true.

Last but not least, responding to the event is done by the client side script. Every control in ASP.NET renders an element to the Browser and that raises the client side events to handle it.

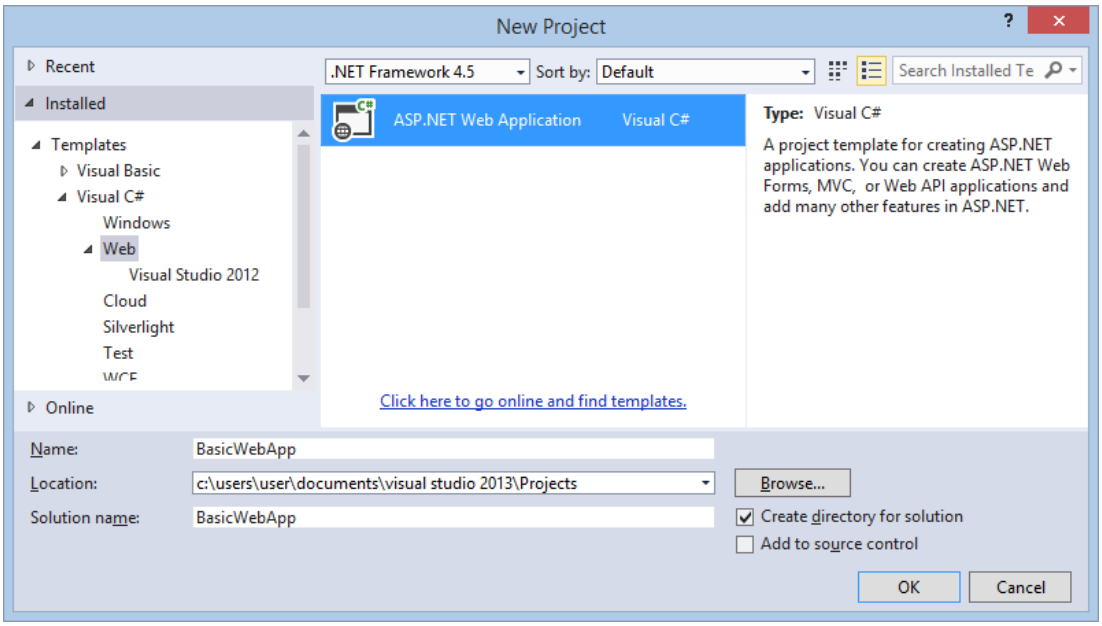
To create a Web application project

1. Open Microsoft Visual Studio.
2. On the **File** menu, select **New Project**.

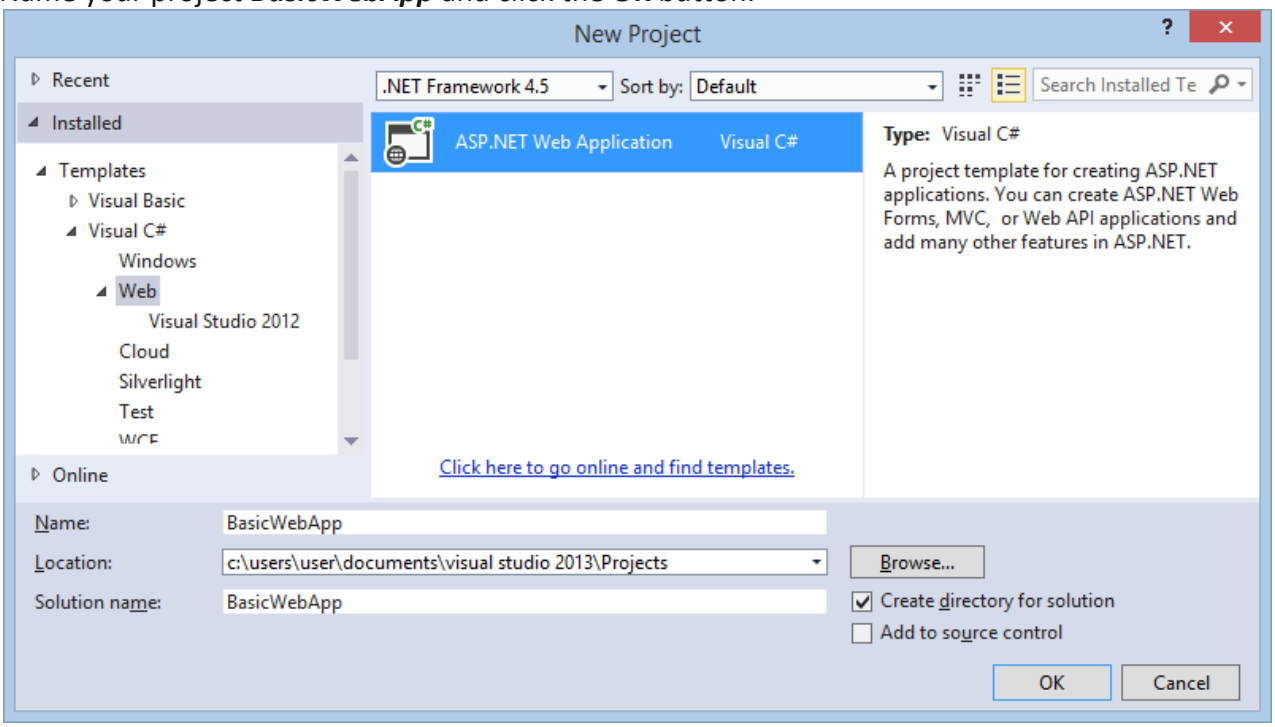


The **New Project** dialog box appears.

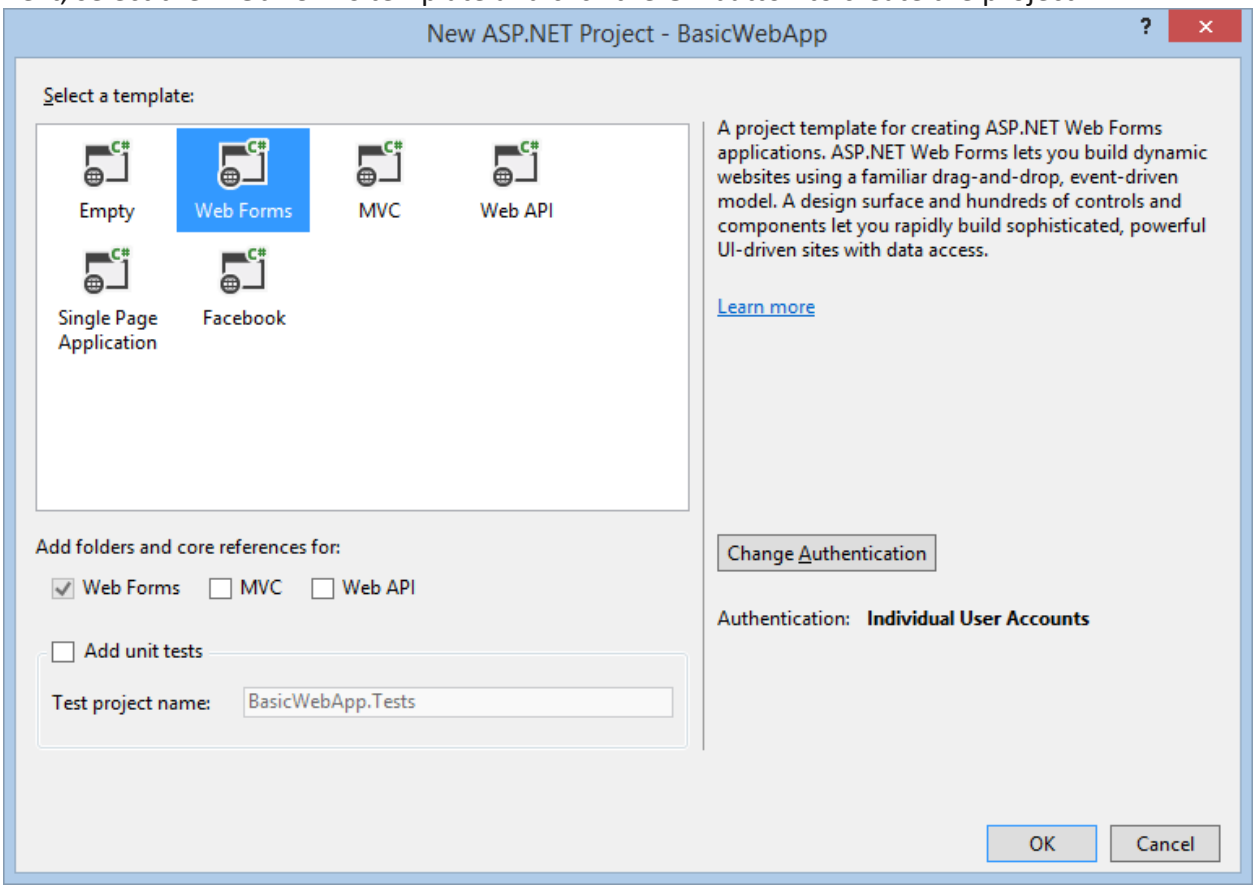
3. Select the **Templates** -> **Visual C#** -> **Web** templates group on the left.
4. Choose the **ASP.NET Web Application** template in the center column.
5. Name your project ***BasicWebApp*** and click the **OK** button.



6. Name your project ***BasicWebApp*** and click the **OK** button.

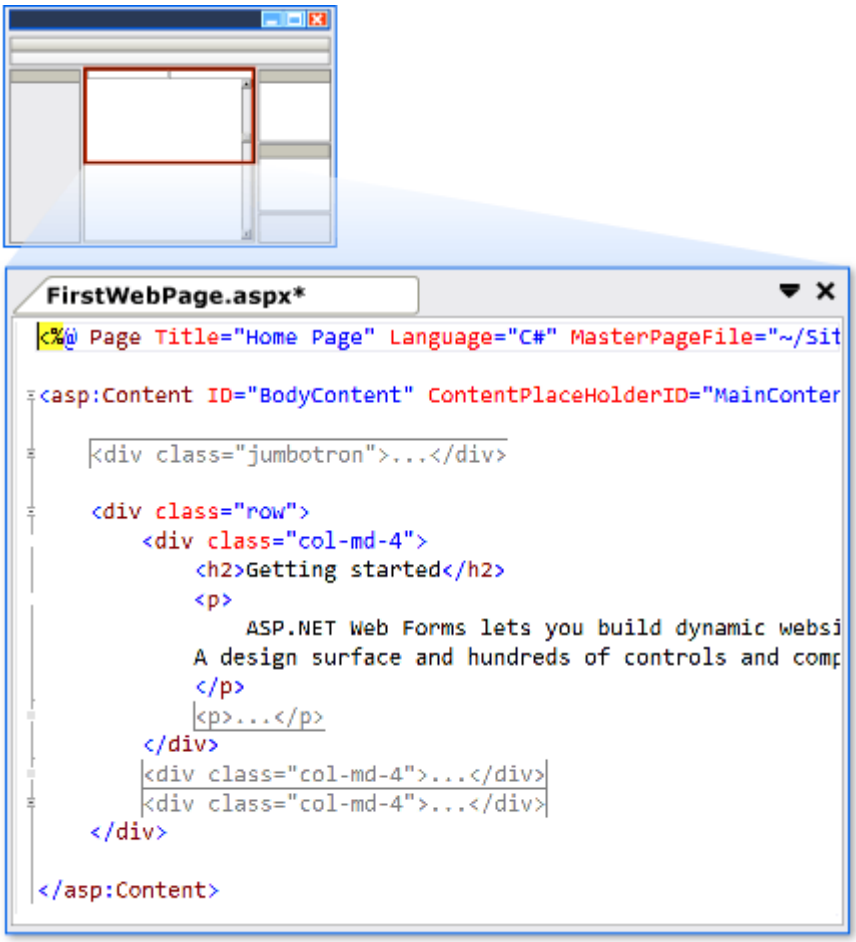


7. Next, select the **Web Forms** template and click the **OK** button to create the project.



Visual Studio creates a new project that includes prebuilt functionality based on the Web Forms template. It not only provides you with a *Home.aspx* page, an *About.aspx* page,

a *Contact.aspx* page, but also includes membership functionality that registers users and saves their credentials so that they can log in to your website. When a new page is created, by default Visual Studio displays the page in **Source** view, where you can see the page's HTML elements. The following illustration shows what you would see in **Source** view if you created a new Web page named *BasicWebApp.aspx*.



Program to design a calculator in C# console based application

```
class Program
  {
    static void Main(string[] args)
    {
      int num1;
      int num2;
      string operand;
      ConsoleKeyInfo status;
      float answer;

      while (true)
      {
        Console.Write("Please enter the first integer: ");
        num1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Please enter the second integer: ");
        num2 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Please enter an operand (+, -, /, *): ");
        operand = Console.ReadLine();

        switch (operand)
        {
          case "-":
            answer = num1 - num2;
            break;
          case "+":
            answer = num1 + num2;
            break;
          case "/":
```

```
                answer = num1 / num2;
                break;
            case "*":
                answer = num1 * num2;
                break;
            default:
                answer = 0;
                break;
        }

        Console.WriteLine(num1.ToString() + " " + operand + " " + num2.ToString() + " = " +
answer.ToString());
        Console.WriteLine("\n\n Do You Want To Break (Y/y)");
        status = Console.ReadKey();
        if(status.Key==ConsoleKey.Y)
        {
            break;
        }
        Console.Clear();
      }
    }
  }
```