MANIPAL INSTITUTE OF TECHNOLOGY

Manipal – 576 104

DEPARTMENT OF COMPUTER SCIENCE & ENGG.

CERTIFICATE

This is to certify that Ms./Mr. …………………...……………………………………

Reg. No. …..…………………… Section: …………… Roll No.: ………………...

has satisfactorily completed the lab exercises prescribed for Internet Technologies Lab [CSE 4111] of Fourth Year B. Tech. Degree in Computer Science and Engg. at MIT, Manipal, in the academic year 2019–2020.

Date: ……...................................

Signature
Faculty in Charge

CONTENTS

Course Objectives

- Acquire in-depth understanding of web application architecture.
- To understand techniques to improve user experience in web applications.
- To gain knowledge about how to interact with database, files and XML.

Course Outcomes

At the end of this course, students will have the

- Ability to develop a basic website using a modern web development tool.
- Ability to design websites with better look and feel.
- Expertise to create real-world web applications.

Evaluation Plan

- Internal Assessment Marks : 60%
  - ✓ Continuous evaluation component (for each experiment):10 marks
  - ✓ The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce.
  - ✓ The marks of the 8 experiments is out of 60 marks.
  - ✓ The total lab internal marks is 60.

- End semester assessment of 2 hour duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre-Lab Session Instructions

1. Be in time and follow the institution dress code.
2. Must Sign in the MS Teams.
3. Make sure to occupy the allotted seat and answer the attendance.
4. Adhere to the rules and maintain the decorum.

In-Lab Session Instructions

- Follow the instructions on the allotted exercises.
- Show the program and results to the instructors on completion of experiments.
- On receiving approval from the instructor, copy the program and results in the Lab Record.
- Prescribed textbooks and class notes can be kept ready for reference if required.

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
    - Comments should be used to give the statement of the problem.
    - Statements within the program should be properly indented.
    - Use meaningful names for variables and functions.

    - Make use of constants and type definitions wherever needed.
    - The website should be well designed and unique.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
    - Solved exercise
    - Lab exercises – to be completed during lab hours.
    - Additional Exercises – to be completed outside the lab or in the lab to enhance the skill.
- In case a student misses a lab class, he/she must ensure that the experiment is completed during the repetition lab with the permission of the faculty concerned but credit will be given only to one day’s experiment(s).
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.
- A sample on how to write the lab observation is provided with this lab note.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO.: 1 **Date:**

C# PROGRAMMING – VARIABLES, OPERATIONS, CONDITIONAL LOGICS, LOOPS, FUNCTIONS

Objectives:

In this lab, student will be able to:

- Understand the fundamentals of C# Programming Language.
- Learn how to use variables, operations, conditional logics, loops & functions.

I. DESCRIPTION

Variables can store numbers, text, dates, times, and they can even point to full-fledged objects. When a variable is declared, it is given a name, and the type of data it will store is specified.

```
//Declare a string variable named myName.
        string myName;
```

Conditional logic - deciding which action to take based on user input, external conditions, or other information.

To build a condition, any combination of literal values or variables along with logical operators can be used.

The comparison operators (<, >, <=, >=) with numeric types and with strings can be used.

C# has three basic types of loops.

- for loop: loop a set number of times
- foreach loop: loop all the items in a collection of data
- while or do…while loop: loop while certain condition holds true

C# provides a foreach block that allows the user to loop through the items in a set of data. Here a variable that represents the required type of data has to be created. The code will then loop until each piece of data in the set has been processed. The variable in the block is read only.

```
string[] stringArray = {“One”, “Two”, “Three”};
foreach (string element in stringArray )
{
    //This code loops 3 times, with element variable set to “One”, then “Two”and then
“Three”
    Console.WriteLine( element + “ ” );   }
```

II. SOLVED EXERCISE:

Steps to create a C# Console application:

1) On the **File** menu, click **New Project**.
   The New Project dialog box appears. This dialog box lists the different types of application that Visual C# Express Edition can create.
2) Select **Console Application** as the project type and change the name of the application to Add_Num. The default location can be changed if required.
3) Click OK.
4) Type the following code in the **Program.cs** file

```
using System;
namespace Add_Num
{
   class Program
   {
      static void Main(string[] args)
      {
         int num1,num2;
         Console.WriteLine("Enter the numbers:");
         string n1 = Console.ReadLine(); // Readline() method returns string type
         string n2 = Console.ReadLine();
         int.TryParse(n1, out num1);
         int.TryParse(n2, out num2);
         int res = num1 + num2;
         Console.WriteLine("The result of addition of {0} and {1} is
         {2}",num1,num2,res);
         Console.Read(); // Read() accepts only single character
      }
   }
}
```

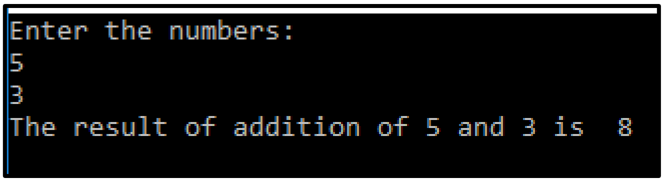
The last line in the program is Console.Read(); which causes the program to pause until ENTER is pressed. If this line is omitted, the command window will close and the user won't be able to see the output of the program.

5) Run the program.

The first program is now complete and ready to compile and run. To do this, either press F5 or click on the **Start** icon in the toolbar.

6) Once the program compiles and runs, the **Console** window opens and the result of addition of two integers is displayed. Press any key to exit the program.

<u>Output:</u>



```
Enter the numbers:
5
3
The result of addition of 5 and 3 is  8
```

Figure 2.1

Steps to create a Windows Forms Application:

1) On the File menu, click New Project.
   The New Project dialog box appears. This dialog box lists the different application types that Visual C# Express Edition can create.
2) Select Windows Forms Application as the project type.
3) Change the name of the application if required.
4) Click OK.
   Visual C# Express Edition creates a new folder for the project with the same name as the project title, and then displays the new Windows Form, titled Form1 in Designer view. The user can switch between the Designer view and Code view at any time by right-clicking on the design surface or code window and then clicking View Code or View Designer respectively.
5) Drag and drop two TextBoxes, two labels and two buttons in the Designer from the ToolBox. Name the buttons as Submit and Clear by right-clicking the buttons and editing their names from the Properties Window.
6) Paste the following code in the Form1.cs file. Check the id's of the various controls in the code and change them according to the control id's. button1_Click and button2_Click are the event handlers for the two buttons in the Designer. These event handlers can be generated by simply double-clicking on the buttons in the Designer View.

```
using System;
namespace WindowsFormsApplication1
```

```
{
   public partial class Form1 : Form
   {
      private void button1_Click(object sender, EventArgs e)
      {
         double n1;
         double.TryParse(TextBox1.Text, out n1);
         TextBox2.Text = (n1 * 2).ToString();
      }
      private void button2_Click(object sender, EventArgs e)
      {
         TextBox1.Text = TextBox2.Text ="";
      }
   }
}
```

7) Build and Run the program.

<u>Output</u>:

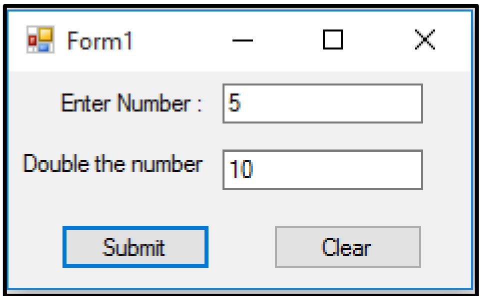


Figure 2.2

III. LAB EXERCISES:

1) Write a simple console application to perform arithmetic operations.
2) Write a Console C# program to realise DateTime.Add member function without using DateTime/TimeStamp instances. The inputs to the program are valid date (in the format "DD: MM: YY: hh:mm:ss") and # of ticks (range from 10000000 - 999999999999) which have to be accepted from the user. The output will be a number which is generated by adding the ticks to the given date. (Note: 1 tick = 100 nano secs)
3) Develop a simple C# windows application to compute the bonus to be paid to an employee based on his performance level using a function. Use TextBox to input

Salary and ComboBox to select performance level. (Performance Level1 = 0.1 * Salary, Level2 to Level4 = 0.09*Salary, Level5 to Level7 = 0.07 * Salary, Level8 to Level10 = 0.05 * Salary)

4) Develop a simple C# windows application to facilitate purchasing order of cars. The form should contain two dropdownlist for car names and colours.It should also contain a listbox for Model and textbox Price. On clicking the “Purchase” button, display the message “ThankYou for purchasing” and on clicking “Cancel”, clear the selections made.

IV. ADDITIONAL EXERCISES:

1) Develop a simple C# windows application to select two types of accessories (Hard Disks and Mobile) using CheckBox controls. Under both categories display the manufacturer names using RadioButtonList controls. When user selects option the corresponding name of the manufacturer should be listed in the label control.
2) Develop a simple C# windows application to display the names of brands of shuttle badminton racquets using label controls. Use TextBox control to accept the brand name and number of racquets the user wants to purchase. Based on the brand selected calculate and display the cost of purchase to the user. (Eg: Cost/Racquet for some brands are: Yonex- Rs2000, Silvers- Rs1000, Cosco-Rs800).

LAB NO.: 2 **Date:**

C# PROGRAMMING – ARRAYS, CLASSES, INHERITANCE AND POLYMORPHISM

Objectives:

In this lab, student will be able to:

- Understand the fundamentals of C# Programming Language.
- Learn how to use arrays, class, inheritance, polymorphism and simple windows controls.

I. DESCRIPTION

Arrays:

Arrays allow you to store a series of values that have the same data type. All arrays start at a fixed lower bound of 0.

```
//Create an array with four strings (from index 0 to index 3).
string[] stringArray = new string[4];
stringArray = {"1", "2", "3", "4"}
//Create a 2 x 4 grid array (with a total of eight integers).
int[,] intArray = new int[2, 4];
intArray = {{1, 2}, {3, 4}, {5, 6}, {7, 8}}
//Access the value in row 0 (first row), column 1 (second column).
int num;
num = intArray[0, 1]  // num is now set to 2.
```

Classes:

Classes are the code definitions for objects. Several instances of a class can be created. Classes interact with each other with the help of three key ingredients:

- Properties: Properties allow the user to access an object’s data. (Some of the properties are read-only. Ex: Length)
- Methods: Methods allow the user to perform an action on an object. (Ex: Open() used to connect to Database)
- Events: Events provide notification that something has happened. (Ex: if a user clicks a button, the Button object fires a Click event.)

Creating a simple class:

```
public class Product
        {
 //Class code goes here
```

```
                    }
```

If there is a class named ‘Product’, then its object can be created as follows:

```
 Product saleProduct = new Product();
```

Product objects can be manipulated in a safe way. This can be done by adding Property Accessors.

Accessors usually have two parts.

- Get accessor - Allows the code to retrieve data from the object.
- Set accessor - Allows the code to set the object’s data.

```
 public class Product
    {
        private string name;
        private decimal price;
        private string imageUrl;
 public string Name
        {
         get
            {
               return name;
            }
         set
           {
               name = value;
           }
   }
}
```

In Main:

```
Product saleProduct = new Product();
saleProduct.Name = "Kitchen Garbage"; // Usage of property
```

Delegates:

Delegates allow to create a variable that “points” to a method. The first step when using a delegate is to define its signature.

The signature is a combination of several pieces of information about a method: its return type, the number of parameters it has, and the data type of each parameter. A delegate variable can point only to a method that matches its specific signature.

```
            private string TranslateEnglishToFrench(string english)
```

```
          {
           //Code here
           }
```
The declaration of Delegate
```
private delegate string StringFunction(string inputString);
```

Creating delegate variable
```
StringFunction  functionReference;
```

functionReference variable can store a reference to the TranslateEnglishToFrench() method

```
functionReference = TranslateEnglishToFrench;
```

Invoke the method using delegate variable
```
string frenchString;
frenchString = functionReference("Hello");
```

Events:

Classes can use the events to allow one object to notify another object that's an instance of a different class. As an illustration, the Product class example has been enhanced with a PriceChanged event that occurs whenever the price is modified through the property procedure. This event won't fire if code inside the class changes the underlying private price variable without going through the property.

//Define a delegate that represents the event.

```
public delegate void PriceChangedEventHandler();
public class Product
{
    // Define the event using the delegate.
    public event PriceChangedEventHandler PriceChanged;
    public decimal Price
    {
        get { return  price; }
        set
        {  price = value;
           //Fire the event , provided there is at least one listener.
```

```
                if(PriceChanged != null)
                 {       PriceChanged();      }
              }
          }
      }
```

To handle an event, a method called an *event handler* is first created. The event handler contains the code that should be executed when the event occurs. Then, the event handler is connected to the event. The event handler would look like a simple method shown here:

```
    public void ChangeDetected()

               {    //This code executes in response to the PriceChanged event

                }
```

The next step is to hook up the event handler to the event.
Use simple assignment statement that sets the event PriceChanged to the event handling method changeDetected by using the += operator

```
      Product saleProduct = new Product("Kitchen", "Garbage", "49.99M")

//This connects the saleProduct.PriceChanged event to an event handling
//procedure called ChangeDetected needs to match the PriceChangedEventHandler
      saleProduct.PriceChanged += ChangeDetected;
//Now the event will occur in response to this code
      salesProduct.Price = saleProduct.Price *2;
```

Static members:

Static properties and methods can be used without a live object. Static members are often used to provide useful functionality related to an object. For example a static variable can be used to keep a count of number of times the object of a class has been instantiated.

To create a static property or method the ***static*** keyword is used after the accessibility keyword

```
    public class TaxableProduct
       {
            public static decimal taxRate = 1.15M;
       }
```

The tax rate information can now be obtained directly from the class, without needing to create an object first:

```
TaxableProduct.taxRate = 1.24M;
```

II. SOLVED EXERCISE:

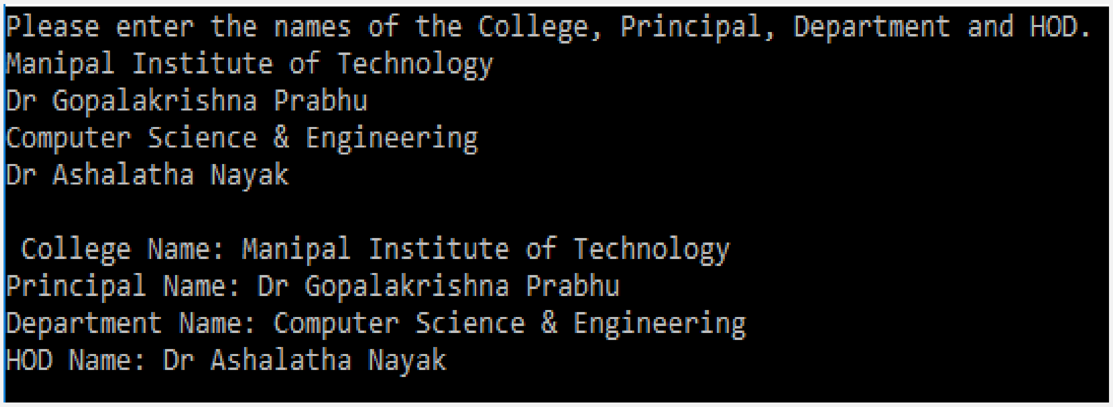
Demonstrate the use of inheritance in C# by developing a simple windows application. Create a base class named ‘College’ with members such as collegeName and principalName and derived class named ‘Department’ with members such as deptName and hodName. Create an object of ‘Department’ class and display the member variables of the object.

Program:

```
namespace Inheritance
{
    class College
    {

        string collegeName;
        string principalName;

        public string CollegeName
        {
            get
            {
                return collegeName;
            }
            set
            {
                collegeName = value;
            }
        }

        public string PrincipalName
        {
            get
            {
                return principalName;
            }
            set
            {
                principalName = value;
            }
        }
    }
  class Department : College
    {
        string deptName;
        string hodName;
        public string DeptName
        {
            get;
            set;
        }

        public string HodName
        {
            get;
            set;
        }
  static void Main(string[] args)
      {
          Department deptObj = new Department();
          Console.WriteLine("Please enter the names of the College, Principal,
Department and HOD.");
          deptObj.CollegeName = Console.ReadLine();
          deptObj.PrincipalName = Console.ReadLine();
          deptObj.DeptName = Console.ReadLine();
          deptObj.HodName = Console.ReadLine();

          Console.WriteLine("College Name: {0}\nPrincipal Name: {1}\nDepartment
Name: {2}\nHOD Name: {3}", deptObj.CollegeName, deptObj.PrincipalName,
deptObj.DeptName, deptObj.HodName);
```

```
        }
    }

}
```

Output:



```
Please enter the names of the College, Principal, Department and HOD.
Manipal Institute of Technology
Dr Gopalakrishna Prabhu
Computer Science & Engineering
Dr Ashalatha Nayak

 College Name: Manipal Institute of Technology
Principal Name: Dr Gopalakrishna Prabhu
Department Name: Computer Science & Engineering
HOD Name: Dr Ashalatha Nayak
```

Figure 3.1

III. LAB EXERCISES:

1. Create a class called "Item" which has the following properties:
   a. Name of the item b. Price of the item

   Create an event in Item class that is fired whenever price changes through the property created. Create a console application that will display the new price of the Item using an EventHandler.
2. Create a console application with class named "Item" which contains one automatic property called "name" of type string and a shared property called "cost" which takes only positive decimal values. Write a function "CalcGst( )" which returns a decimal value (Formula for GST:8% of the base cost).
3. Write a program in C# to demonstrate declaration, instantiation, and use of a delegate, called TrafficDel and a class called TrafficSignal with the following delegate methods Yellow() ,Green() and Red() which prints what each signal is meant for.

IV. ADDITIONAL EXERCISES:

1) Write a program in C# to demonstrate declaration, instantiation, and use of a delegate, TaxCalculator, that can be used to reference methods, TaxIndia (decimal) and TaxUS (decimal), which calculates tax for salaried individuals based on various tax slabs according to government rules of the particular nation and returns the tax payable value in decimal.
2) Create a class named EBBill. This class contains four variables named ownerName, houseNumber, unitsConsumed and metreRent. Among these four variables, the variable metreRent is a shared variable, while other three variables are instance variables. Create simple C# windows application to compute electricity bill and display the name of the owner and electricity bill. (Total amount = 1.2 * Units Consumed + Metre Rent).

LAB NO. : 3 **Date:**

wEB FORMS AND WEB CONTROLS

Objectives:

In this lab, student will be able to:

- Understand the fundamentals of web forms creation
- Design ASP.NET web applications using various web controls

I. DESCRIPTION

ASP.NET websites include multiple web pages. Server controls are considered to be the basic building block of every web form. Using these server controls, ASP.NET applications (a combination of files, pages, handlers, modules and executable code that can be invoked from a virtual directory on a web server) are created.

Server controls are of 2 types.

1. HTML Server Controls

   These provide an object interface for standard HTML elements. These controls are ideal if the user is a seasoned web programmer who prefers to work with familiar HTML tags (at least at first). They are also useful when migrating ordinary HTML pages or classic ASP pages to ASP.NET, because they require the fewest changes. The normal HTML tags with attribute runat=”server” and id are considered as HTML server controls.

   Eg: <input type = submit value = “OK” ID = “Click” runat = “server” />

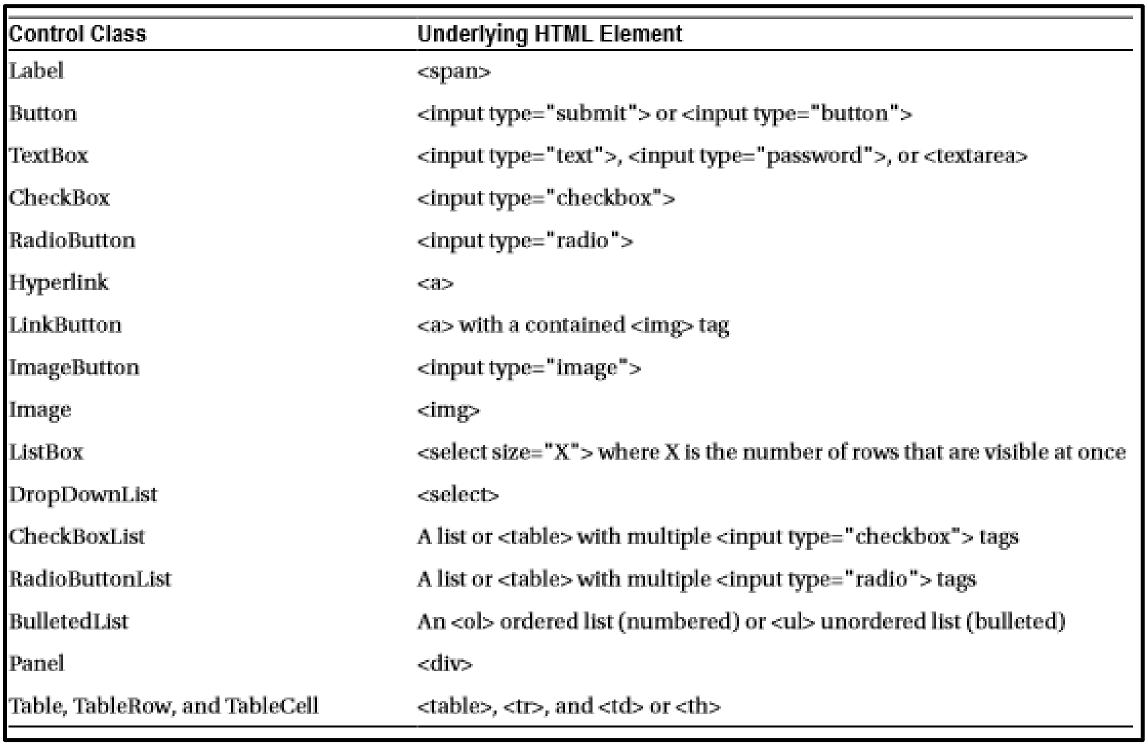
2. Web Server Controls

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

Eg: <asp: Button ID = “Click” runat = “server” Text = “Click” Onclick = “Button_click” /> //Definition for Button_click (Event handler) is written in .cs file.

The file types that end with .aspx are ASP.NET web pages. They contain the user interface and, optionally, the underlying application code. Users request or navigate directly to one of these pages to start the web application.

The file types that end with .cs are code-behind files that contain C# code. They allow the user to separate the application logic from the user interface of a web page. The web.config is the configuration file for the ASP.NET application. It includes settings for customizing security, state management, memory management, and much more. The basic web control classes and the underlying HTML elements are shown in the table below:



Control Class	Underlying HTML Element
Label	<span>
Button	<input type="submit"> or <input type="button">
TextBox	<input type="text">, <input type="password">, or <textarea>
CheckBox	<input type="checkbox">
RadioButton	<input type="radio">
Hyperlink	<a>
LinkButton	<a> with a contained <img> tag
ImageButton	<input type="image">
Image	<img>
ListBox	<select size="X"> where X is the number of rows that are visible at once
DropDownList	<select>
CheckBoxList	A list or <table> with multiple <input type="checkbox"> tags
RadioButtonList	A list or <table> with multiple <input type="radio"> tags
BulletedList	An <ol> ordered list (numbered) or <ul> unordered list (bulleted)
Panel	<div>
Table, TableRow, and TableCell	<table>, <tr>, and <td> or <th>

Figure 4.1 Web Control classes and HTML elements

Procedure to generate a web form in Visual studio

1. Right-click the website in the Solution Explorer and choose Add ➤ Add New Item.

2. In the “Add New Item” dialog box, choose Web Form (which should be first in the list).

3. Type a name for the new page (such as CurrencyConverter.aspx).

4. Make sure the “Place Code in Separate File” option is selected (it’s in the bottom-right corner of the window).

5. Click Add to create the page.

The web form begins with a page directive that provides ASP.NET basic information about how to compile the web page. It indicates the language the user is using for the code and the way the user has connected the event handlers. If the user is using the code-behind approach, which is recommended, the page directive also indicates where the code file is located and the name of the custom page class.

The user needs to add the attribute runat = "server" to each tag that he/she wants to transform into a server control. The user should also add an ID attribute to each control that needs to interact with in code. The ID attribute assigns the unique name that can be used to refer to the control in code.

II. SOLVED EXERCISE:

Develop a currency converter using the constructs of ASP.NET web forms and web controls.

CurrencyConverter.aspx

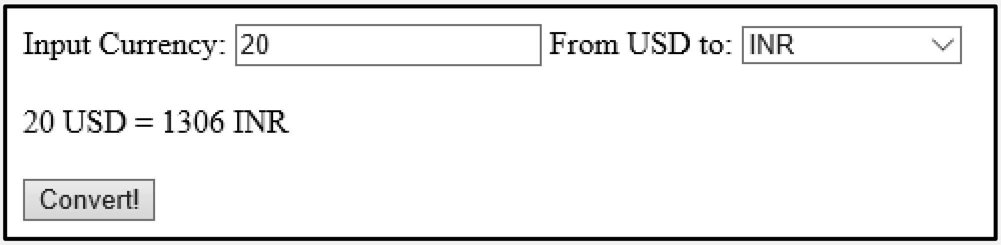
```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="CurrencyConverter.aspx.cs" Inherits="CurrencyConverter" %>

<!DOCTYPE html>
```

```
<html >
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
   <asp:Label runat="server" id=”lb” >Input Currency:/>
      <asp:TextBox runat="server" id="inputCurrency"/>
      < asp:Label  id="fromLable" runat="server" >From USD to:/>
      <asp:DropDownList id="fromCurrencyDropDown" runat="server"/>
      <br /> <br />
            < asp:Label  runat="server" id="labelAns" />
      <br /> <br />
      <asp:Button Text="Convert!" OnClick="SubmitClickedEvent" runat="server"/>
      <br />
      <br />
   </div>
   </form>
</body>
</html>
```

<u>CurrencyConverter.aspx.cs</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class CurrencyConverter : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      if (!IsPostBack)
      {
            fromCurrencyDropDown.Items.Add(new ListItem("INR", "65.3"));
            fromCurrencyDropDown.Items.Add(new ListItem("Japanese Yen", "110.33"));
            fromCurrencyDropDown.Items.Add(new ListItem("Euro", "0.85"));
        }
    }

    protected void SubmitClickedEvent(object sender, EventArgs e)
    {
        string convertedValue = convertCurrency();
        labelAns.InnerText = convertedValue.ToString();
    }

    string convertCurrency()
    {
        double oldValue = 0;
        double newValue = 0;
        double.TryParse(inputCurrency.Value, out oldValue);

        ListItem temp =
fromCurrencyDropDown.Items[fromCurrencyDropDown.SelectedIndex];
        newValue = oldValue * double.Parse(temp.Value);

        string s = oldValue.ToString() + " USD = " + newValue.ToString() + " " +
temp.Text;
                return s;   }}
```

Output


Input Currency: 20 From USD to: INR

20 USD = 1306 INR

Figure 4.2

III. LAB EXERCISES:

1) Develop a simple ASP.NET web form that generates the front cover for a magazine. The form should provide the options for selecting the image, background color, changing font size, color etc. Input messages must be taken from the user so as to display it on the front cover with legible font family and font size. The controls added must be proportionate to the web page size.
2) Create a web form using web controls, with employee ids in dropdown list. When the user selects the employee id, his image should be displayed in the image control. Let the next focus be on the TextBox named “name of the employee”. Use another TextBox for date of joining. Add a button as “Am I Eligible for Promotion”. On clicking the button, if he has more than 5 years of experience, then display “YES” else “NO” in the label control.
3) Develop a simple dynamic e-card generator. E-card generator should provide options to select background colour, type of font, font size. An image can be included in the e-card with an option to provide input message.
4) Develop simple application using web controls to reproduce the given Captcha. Upon match, suitable message has to be displayed. If there is a mismatch for more than 3 times, TextBox has to be disabled. (Hint: Use Hidden TextBox).

IV. ADDITIONAL EXERCISES:

1) Create a web form which allows the teacher to enter the student details like name, date of birth, address, contact number, email id and marks of English, Physics and Chemistry. After filling all the information and on clicking submit button, details should be added to a textarea displayed on the same page. Display the total percentage of marks obtained in a label.
2) Create an aspx and corresponding .cs file to design a pamphlet on test drive of bikes, by taking inputs from user through web controls such as textarea, checkBox, buttons, labels and images (any other web controls apart from these can also be added as per requirement).

LAB NO.: 4 **Date:**

STATE MANAGEMENT

Objectives:

In this lab, student will be able to:

- Learn the various state management techniques.
- Develop ASP.NET web applications with state management techniques.

I. DESCRIPTION

The most significant difference between programming for the Web and programming for the desktop is state management—how you store information over the lifetime of your application.. This information can be as simple as a user's name or as complex as a stuffed-full shopping cart for an e-commerce store.

In a typical web request, the client connects to the web server and requests a page. When the page is delivered, the connection is served, and the web server discards all the page objects from memory. By the time the user receives a page, the web page code has already stopped running, and there's no information left in the web server's memory.

The client side state management techniques include viewstate and cookies. The session state and application state are considered to be a server side state management technique. Cross page posting and querystring are the techniques used for transferring information between web pages.

Client Side State Management

Client side objects are page(s) scope objects that used to keep data in safe from postbacks, or to be transferred between pages.

1. <u>Viewstate</u>
    - The most common ways to store information is in view state.
    - Uses a hidden field that ASP.NET automatically inserts in the final, rendered HTML of a web page.
    - The ViewState property of the page provides the current view state information.
    - The "this" keyword refers to the current Page object. It's optional.

this.ViewState[“Counter”] = 1;

- The key name is used to retrieve the value. The user needs to cast the retrieved value to the appropriate data type using the casting syntax.
- This extra step is required because the ViewState collection stores all items as basic objects, which allows it to handle many different data types.

int counter;

counter = (int)this.ViewState[“Counter”];

2. Cookies
    - Cookies provide another way so that the user can store information for later use.
    - They are small files that are created in the web browser’s memory (if they’re temporary) or on the client’s hard drive.
    - Both the Request and Response objects (which are provided through Page properties) provide a Cookies collection.
    - The user can retrieve cookies from the Request object, and set cookies using the Response object.
    - To set a cookie, just create a new HttpCookie object.

    HttpCookie cookie = Request.Cookies[“Preference”];

    - Set the value for Cookie

    cookie["LanguagePref”] = "English" ‘

    - Adding values to the Cookie

    cookie["Country”] = "US" ;

    - Attach Cookie to the current web response

    Response.Cookies.Add(cookie);

    - A cookie added in this way will persist until the user closes the browser and will be sent with every request.
    - Creating a longer-lived cookie is by setting an expiration date.

```
cookie.Expires = DateTime.Now.AddYears(1);
```

- Retrieve cookies by cookie name using the Request.Cookies collection.

```
HttpCookie cookie = Request.Cookies["Preferences”];
```

- Check to see whether a cookie was found with this name.

```
string language;
if(cookie ! = null)
        {
                language = cookie[“LanguagePref”];
         }
```

- The only way to remove a cookie is by replacing it with a cookie that has an expiration date that has already passed.

```
HttpCookie cookie = new HttpCookie(“Preferences”);
cookie.Expires = DateTime.Now.AddDays(-1) ;
Response.Cookies.Add(cookie) ;
```

3. Cross page posting (Transfer information between pages)
   - A cross-page postback is a technique that extends the postback mechanism, so that one page can send the user to another page, with the complete information of that page.
   - The property that supports cross-page posting is PostBackUrl set to the name of another web form.
   - When the user clicks the button, the page will be posted to that new URL with the values from all the input controls on the current page.
   - Example—A page named “CrossPage1.aspx” that defines a form with two text boxes and a button. When the button is clicked, it posts to a page named “CrossPage2.aspx”.

```
    <asp:Button runat="server" ID="cmdPost"
PostBackUrl="CrossPage2.aspx" Text="Cross-Page Postback" />
```

   - To get more specific details, such as control values, you need to cast the PreviousPage reference to the appropriate page class.

```
 public partial class CrossPage2 : System.Web.UI. Page

 {
```

```
protected void Page_Load(Object sender, EventArgs e)
  {
    if( PreviousPage != null)
     {
       lblInfo.Text = “You came from a page titled ”+PreviousPage.Title;
      }
  }
}
```

- The page checks for null reference before attempting to access the PreviousPage object. If it’s a null reference, no cross-page postback took place.

4. <u>Querystring</u> (Transfer information between pages)

- These are used to transfer the data through pages.
- Common approach is to pass information using a query string in the URL.
- This approach is commonly found in search engines.

  http://www.google.co.in/search?q=organic+gardening
- The querystring is the portion of the URL after the question mark. In this case, it defines a single variable named q, which contains the string organic+gardening.

  Go to newpage.aspx. Submit a single query string argument named recordID, and set to 10.

  ```
  Response.Redirect("newpage.aspx?recordID=10")
  ```
- The user can send multiple parameters as long as they’re separated with an ampersand (&):

  Go to newpage.aspx. Submit two query string arguments: recordID (10) and mode (full).

  ```
  Response.Redirect("newpage.aspx?recordID=10&mode=full")
  ```
- It can receive the values from the QueryString dictionary collection exposed by the built-in Request object:

  ```
  string ID = Request.QueryString["recordID”];
  ```
- It supports all characters that are alphanumeric or one of a small set of special characters (including $-_.+!*'(),).
- Some characters have special meaning.
  - The ampersand (&) is used to separate multiple query string parameters
  - The plus sign (+) is an alternate way to represent a space

  - The number sign (#) is used to point to a specific bookmark in a web page.
- If the user tries to send query string values that include any of these characters, some of the data will be lost.
- To solve this problem **URL encoding** is applied on text values before they are placed in the query string.
- With URL encoding, special characters are replaced by escaped character sequences starting with the percent sign (%), followed by a two-digit hexadecimal representation.

  Example: The & character becomes %26
- To perform URL encoding, the UrlEncode() and UrlDecode() methods of the HttpServerUtility class are used.

```
string url= "QueryStringRecipient.aspx?"
Url += "Item="  + Server.UrlEncode(lstItems.SelectedItem.Text) + "&“;
Url += "Mode=" + chkDetails.Checked.ToString() ;
Response.Redirect(Url)
```
- Use the UrlDecode() method to return a URL-encoded string to its initial value.
- ASP.NET automatically decodes the values when they are accessed through the Request.QueryString collection in query strings.

Storing custom objects in viewstate

To store an item in view state, ASP.NET must be able to convert it into a stream of bytes so that it can be added to the hidden input field in the page. This process is called **serialization**.

To make the objects serializable, a Serializable attribute has to be added before the class declaration.

```
[Serializable]
```

Then it can be stored in View State.

```
Customer cust  = new Customer("Marsala", "Simons")
ViewState("CurrentCustomer") = cust
```

It is needed to cast the view state object.

```
//Retrieve a customer from view state.
   Customer cust;
```

cust = (Customer)ViewState[“CurrentCustomer”]

Server Side State Management

Server side objects are domain scope objects that are used to store the data for the use of whole application, including business logic and script language.

1. Session State
    - It allows the user to store any type of data in memory on the server.
    - The information is protected, because it is never transmitted to the client, and it’s uniquely bound to a specific session.
    - Every client that accesses the application has a different session and a distinct collection of information.
    - ASP.NET tracks each session using a unique 120-bit identifier.
    - ASP.NET uses a proprietary algorithm to generate this value.
    - This ID is the only piece of session-related information that is transmitted between the web server and the client.
    - When the client presents the session ID, ASP.NET looks up the corresponding session, retrieves the objects you stored previously, and places them into a special collection so they can be accessed in the code.
    - This process takes place automatically.
    - For this system to work, the client must present the appropriate session ID with each request.
    - The user can accomplish this in two ways:
        - *Using cookies*: In this case, the session ID is transmitted in a special cookie (named ASP.NET_SessionId), which ASP.NET creates automatically when the session collection is used. This is the default.
        - *Using modified URLs*: In this case, the session ID is transmitted in a specially modified (or *munged*) URL. This allows the user to create applications that use session state with clients that don’t support cookies.

- The user can interact with session state using the System.Web.SessionState.HttpSessionState class, which is provided in an ASP.NET web page as the built-in Session object.
- The syntax for adding items to the collection and retrieving them is basically the same as for adding items to a page's view state.
- For example, the user might store a DataSet in session memory like this:

  ```
  Session["InfoDataSet“] = dsInfo;
  ```

- The user can then retrieve it with an appropriate conversion operation:

  ```
  dsInfo = (DataSet)Session["InfoDataSet“];
  ```

- The user can configure session state through the web.config file for the current application.
- The configuration file allows to set advanced options such as the timeout and the session state mode.

2. Application State
    - Application state allows to store global objects that can be accessed by any client.
    - Application state is based on the System.Web.HttpApplicationState class, which is provided in all web pages through the built-in Application object.
    - Application state supports the same type of objects, retains information on the server, and uses the same dictionary-based syntax.
    - Application state items are stored as objects, so the user needs to cast them when retrieving them from the collection.
    - Items in application state never time out. They last until the application or server is restarted or the application domain refreshes itself.

II. SOLVED EXERCISE:

Develop a counter application using state management technique.

Count.aspx

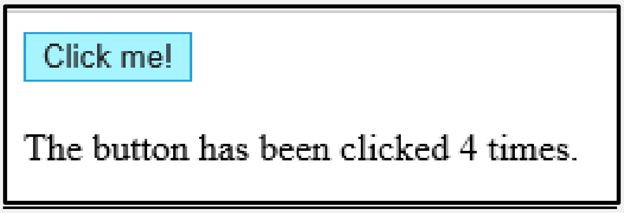
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Count.aspx.cs"
Inherits="Count" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Counter</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Button ID="buttonCounter" runat="server" OnClick="buttonCounter_Click"
Text="Click me!" />
      <br /><br />
      <asp:Label ID="labelCounter" runat="server" />
   </div>
   </form>
</body>
</html>
```

Count.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Count : System.Web.UI.Page
{
   protected void buttonCounter_Click(object sender, EventArgs e)
   {
      int counter;
      if(ViewState["count"] == null)
```

```
          counter = 1;
        else
         counter = (int)ViewState["count"] + 1;

        ViewState["count"] = counter;
        labelCounter.Text = "The button has been clicked " + counter.ToString() + "
times.";
    }
}
```

Output



The button has been clicked 4 times.

Figure 5.1

III. LAB EXERCISES:

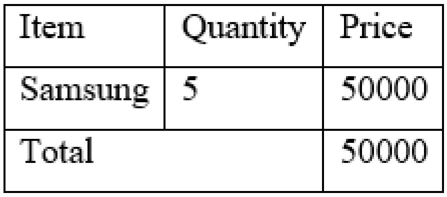
1) Develop a simple ASP.NET application to demonstrate use of QueryString to transfer information between pages. User should be presented a list of entries [car manufacturers] using dropdown list and text box which contains model name of the manufacturer. When the user chooses an item by clicking the appropriate manufacturer in the list, the user is forwarded to a new page. This page displays the car manufacturer name and the model name. Try to avoid potential risk by using URL encoding technique.
2) Create a page firstPage.aspx with two TextBoxes [Name, Roll], DropDownList [Subjects], and a button. Create another page secondPage.aspx with two labels and button. When the user clicks the button in first Page he should be sent to the second page and display the contents passed from first page in the label1. Use the session variable. When the user clicks the button in second page, the counter value should display in the label2. For each click the counter value should increment by 1. Store the counter value using cookie.
3) Develop a simple ASP.NET application to store a customer name and shopping cart inside a cookie. Demonstrate the use of cookie by making the shopping cart,

and clicking the "ContinueShopping" button. Then, close the browser, and request the page again. The second time, the page should find the cookie, display the name of the items in the shopping cart, and display a welcome message with customer name.

4) Develop simple login page using web controls. Upon login, display the global counter that tracks the number of times users have successfully logged in. If login fails more than 3 times, disable the TextBox. (Hint: Use Session and Application variables)

IV. ADDITIONAL EXERCISES:

1) Create a Register page and Success page with the following requirements:
   i. Register page should contain four input TextBoxes for UserName, Password, Email id and Contact Number and also a button to submit. Make the username as compulsory field.
   ii. On button click, Success page is displayed with message "Welcome {UserName}" and also his Email and Contact Number has to be displayed.
   iii. On button click without entering Username, an error message should be displayed to the user in same page.
   iv. Use secure technique to send details to the Success page.
2) Design a website with two pages.
   First page contains:
   Dropdown list with HP, Nokia, Samsung, Motorola, Apple as items.
   CheckBox with Mobile and Laptop as items.
   TextBox to enter quantity.
   There is a button with text as "Produce Bill".
   On Clicking Produce Bill button, item should be displayed with total amount on another page.



Item	Quantity	Price
Samsung	5	50000
Total		50000

Figure 5.2

LAB NO.: 5 **Date:**

VALIDATION, THEMES AND MASTER PAGES

Objectives:

In this lab, the student will be able to:

- Understand how to use the validation controls in an ASP.NET web page and how to get the most out of them with sophisticated regular expressions and custom validation functions.
- Learn about themes and master pages, which can ensure that all the pages on the website share a standardized look and layout.

I. DESCRIPTION

ASP.NET validation controls validate user input to ensure that useless, unauthenticated, or contradictory data do not get stored.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- Validation Summary

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to ensure that the textbox is not empty.

```
<asp:RequiredFieldValidator ID="rfvcandidate"
  runat="server" ControlToValidate ="ddlcandidate"
  ErrorMessage="Please choose a candidate"
  InitialValue="Please choose a candidate">
 </asp:RequiredFieldValidator>
```

The RangeValidator control verifies that the input value falls within a predetermined range.

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
```

```
ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

```
<asp:CompareValidator ID="vldRetype" runat="server"
ErrorMessage="Your password does not match." ControlToCompare="txtPassword"
ControlToValidate="txtRetype" >
</asp:CompareValidator>
```

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

```
<asp:RegularExpressionValidator id="vldEmail" runat="server"
ErrorMessage="This email is missing the @ symbol."
ValidationExpression=".+@.+" ControlToValidate="txtEmail" />
```

The CustomValidator control allows writing application-specific custom validation routines for both the client side and the server side validation.

The client-side validation is accomplished through the ClientValidationFunction property. The client-side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

<asp:CustomValidator ID="vldCode" runat="server" ErrorMessage="Try a string that starts with 014."ValidateEmptyText="False" ControlToValidate="txtCode" onServerValidate="vldCode_ServerValidate" />

Code for the server side validation event:

```
protected void vldCode_ServerValidate(Object source, ServerValidateEventArgs e)
```

```
{
try
{
        // Check whether the first three digits are divisible by seven.
        int val = Int32.Parse(e.Value.Substring(0, 3));
        if (val % 7 == 0)
        {
        e.IsValid = true;
        }
       else
        {
        e.IsValid = false;
}
}
catch
{
        // An error occurred in the conversion.
        // The value is not valid.
        e.IsValid = false;
}
}
```

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- ShowSummary : shows the error messages in the specified format.
- ShowMessageBox: shows the error messages in a separate window.

Themes:

To use the theme in a web application, the user needs to create a folder that defines it. The created folder should be placed in the App_Themes folder placed inside the top-level directory for the web application. An application can contain definitions for multiple themes, but each theme should be in the separate folder. Only one theme can be active on a given page at a time.

To make the theme accomplish the work, the creation of at least one skin file in the theme folder is necessary. A skin file is a text file with the .skin extension. Skin file is essentially a list of control tags; the control tags do not need to completely define the control. Set the properties that need to be standardized.

To add a theme to the project, select Website ➤ Add New Item and choose Skin File.

Master Pages:

ASP.NET master pages allow the user to create a consistent layout for the pages in the application. A single master page defines the look and feels and standard behavior that the user wants for all of the pages (or a group of pages) in his application. The user can then create individual content pages that contain the content he wants to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

The master page is identified by a special @ Master directive that replaces the @ Page directive that is used for ordinary .aspx pages.

In addition to static text and controls that will appear on all pages, the master page also includes one or more ContentPlaceHolder controls. These placeholder controls define regions where replaceable content will appear. In turn, the replaceable content is defined in content pages.

Sample Master Page

```
<%@ Master Language="C#" CodeFile="MasterPage.master.cs"
Inherits="MasterPage"%>
<html>
<head runat="server" >
   <title>Master page title</title>
</head>
<body>
   <form id="form1" runat="server">
      <table>
        <tr>
           <td><asp:contentplaceholder id="Main" runat="server" /></td>
```

```
            <td><asp:contentplaceholder id="Footer" runat="server" /></td>
          </tr>
        </table>
    </form>
</body>
</html>
```

Sample Content Page

```
<%@ Page Language="C#" MasterPageFile="~/Master.master" Title="Content Page1"
%>
<asp:Content ID="Content1" ContentPlaceHolderID="Main" runat="Server">
    Main content.
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="Footer" runat="Server" >
    Footer content.
</asp:content>
```

II. SOLVED EXERCISE:

Create two themes: Monochrome and DarkGrey. For each theme, add the CSS layout, which is applied to the site automatically. Configure the application to use one of the themes and then switch to the other to see the differences.

MainPage.aspx contents:

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

CodeFile="MainPage.aspx.cs" Inherits="MainPage" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:DropDownList ID="themeDropDown" runat="server"
        AutoPostBack="true"
        OnSelectedIndexChanged="themeDropDown_SelectedIndexChanged">
        </asp:DropDownList>
        <asp:Label ID="Label1" runat="server" Text="ThemeDemo"></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server" Text=" "></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Click Me" />
     </div>
     </form>
 </body>
 </html>

MainPage.aspx.cs contents:
using System;
public partial class MainPage : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
         {

            themeDropDown.Items.Add("Monochrome");
            themeDropDown.Items.Add("DarkGrey");
            if (Session["Theme"] != null)
            {
             //FindByText searches the DropDownList for an Item with a Text property
             that equals the specified text.
             themeDropDown.Items.FindByText(Session["Theme"].ToString()).Selecte
             d = true;
            }
         }
    }

 protected void Page_PreInit(object sender, EventArgs e)//change the theme
 {
        if (Session["Theme"] != null)
         {
            // sets the theme of the page dynamically to selected value from the
            DropDownList
            Page.Theme = Session["Theme"].ToString();
         }
  }
protected void themeDropDown_SelectedIndexChanged(object sender, EventArgs
e)
   {
          Session["Theme"] = themeDropDown.SelectedValue;
          Server.Transfer(Request.FilePath);
   }
}

Monochrome.skin contents:
<asp:Label runat="server" Font-Bold="true" Font-Names="Comic Sans MS"
ForeColor="BlueViolet" Font-Size="Large" />
<asp:TextBox  runat="server"  Font-Bold="true"  Font-Names="Comic Sans MS"
ForeColor="BlueViolet" Font-Size="Large" />
<asp:Button runat="server"   Font-Bold="true" Font-Names="Comic Sans MS"
ForeColor="BlueViolet" Font-Size="Large" />

Darkgrey.skin contents:
<asp:Label    runat="server"    Font-Names="Times New Roman"
ForeColor="DarkGray" Font-Size="Small" Font-Italic="true" />
<asp:TextBox runat="server" Font-Names="Times New Roman"
ForeColor="DarkGray" Font-Size="Small" Font-Italic="true" />
<asp:Button runat="server" Font-Names="Times New Roman"
ForeColor="DarkGray" Font-Size="Small" Font-Italic="true"/>
```

III. LAB EXERCISES:

1) Design a Student House President Election 2017 form with fields such as Choose a candidate (use DropDownList), House (use RadioButtonList), Class (6th to 12th), email id, parent contact number and a submit button. Display message “Successfully Submitted” on clicking Submit button i.e. after successful validation of election form. Use:

1. RequiredFieldValidator - for Choose Candidate, House, Class, Email Id fields.
2. RangeValidator – for Class field.
3. RegularExpressionValidator - for Email field.
4. CustomValidator - for contact number field.
5. Validation Summary to show errors in a message box

2) Develop a Web Application using ASP.NET for an E-Commerce firm. The master page should consist of name of the firm, Logo and contact details. Also, it should provide hyperlinks to Electronics, Baggages and Offers zone. These three pages should be designed as content pages. The hyperlinks should navigate to these content pages associated with Master Page designed. The Electronics page should display the categories namely mobiles, laptops and printers. Also display the vendor names for all the categories in DropDownList controls. The Baggages page can have images of laptop bags, trolley bags, and backpacks. In the Offers zone page, add images to display at least two offers. All the pages must have the same background image (something that represents the firm). All the pages should be neat and well designed (the contents added must be proportional to page size) and have an attractive look.

3) Create two themes: Summer and Monsoon. For each theme, add the CSS layout, which is applied to the site automatically. Use appropriate background images to represent each theme. Configure the application to use one of the themes and then switch to the other to see the differences. Extend the above-created themes by adding a DropDownList control which contains the available themes so that a user can choose to dynamically switch between the themes. (any other appropriate controls(e.g., TextBox) can be added to the page to show the effect of themes changing).

4) Create a web application which includes Master Page and two content pages. The master page should contain header footer and left pane. Include the stylesheet file for the master page formatting. In the left pane, there should be a label. It should display the Mobile Name selected from the drop-down list box of the First content page. The first content page should contain dropdown list box with different mobile names, and button. Some theme should be set as default. When you click the button second content page should be displayed. The second content page should contain the label where selected mobile's specification (Screen, Memory, Camera, and Battery) should be displayed. There should be a back button which when clicked will display the first content page.

IV. ADDITIONAL EXERCISES:

1) Create a Webform which contains a dropdown list, TextBox, and a button. Dropdown list contains options like Email, Phone, and UserId. On click of the "validate" button, the value in the TextBox should be validated using custom validator based on what is selected in the dropdown. (Email should contain a @ symbol, Phone should contain only numbers and UserId should only contain capital letters)
2) Create a web application with a Master page having a Panel and a Label (My Page), two buttons (Internal, External) and a DropDownList with values- Please select a theme, Solid and Dotted. On selecting “Solid,” apply changes to panel width, border, and background color. Also, change the button background and foreground colors. Apply similar changes on the selection of “Dotted” theme. On click of Internal and External buttons navigate to respective content pages with internal style and external styles applied to a label (change font family, font size, color, and font weight).

LAB NO.: 6 **Date:**

WORKING WITH DATA - I

Objectives:

In this lab, the student will be able to:

- Understand the ADO.NET fundamentals.
- Connect the Web pages to the SQL Server database using direct & disconnected data access methods.

I. DESCRIPTION

ADO.NET is the technology that .NET applications use to interact with a database. Most of the Web Applications use a full relational database management system (RDBMS), such as SQL Server. Almost every piece of software ever written works with data. A typical web application is often just a thin user interface shell on top of sophisticated data-driven code that reads and writes information from a database.

Process for connecting to the database / creating a database in an SQL server

There is a tab for the Server Explorer on the right side of the Visual Studio window, grouped with the Toolbox and collapsed. Click the tab to expand it. If not, choose View ➤ Server Explorer to show it (or View ➤ Database Explorer in Visual Studio Express for Web). Using the Data Connections node in the Server Explorer, the user can connect to existing databases or create new ones.

1. Right-click the Data Connections node and choose Add Connection.

2. When the “Choose Data Source” window appears, select Microsoft SQL Server and then click Continue.

3. If the user is using a full version of SQL Server, enter “localhost” as the server name. This indicates that the database server is the default instance on the local computer. (Replace this with the name of a remote computer if needed.) If the user is using SQL Server Express LocalDB (the version that’s included with Visual Studio), then enter (localdb)\MSSQLlocalDB instead of localhost. If it is SQL Server Express, the user needs to enter localhost\SQLEXPRESS instead.

4. Click the Test Connection button to verify if the location of the database is correct. If the user has not installed a database product yet, this step will fail. Otherwise, the database server is installed and running.

5. In the Select or Enter a Database Name list, choose the required database. If the user wants to see more than one database in Visual Studio, he needs to add more than one data connection.

6. Click OK. The database connection appears in the Server Explorer window. The user can now explore its groups to see and edit tables, stored procedures, and more. For example, if the user right-clicks a table and choose Show Table Data, a grid of records are seen that can be browsed and edited.

SQL Basics

When working with ADO.NET, however, the user will probably use only the following standard types of SQL statements:

- A Select statement retrieves records.
  SELECT column_name FROM table_name;
- An Update statement modifies existing records.
  UPDATE table_name SET some_column = some_value WHERE some_column = some_value;
- An Insert statement adds a new record.
  INSERT INTO table_name (column1, column2) VALUES (value1, ‘value2’);
- A Delete statement deletes existing records

DELETE FROM table_name WHERE some_column = some_value;

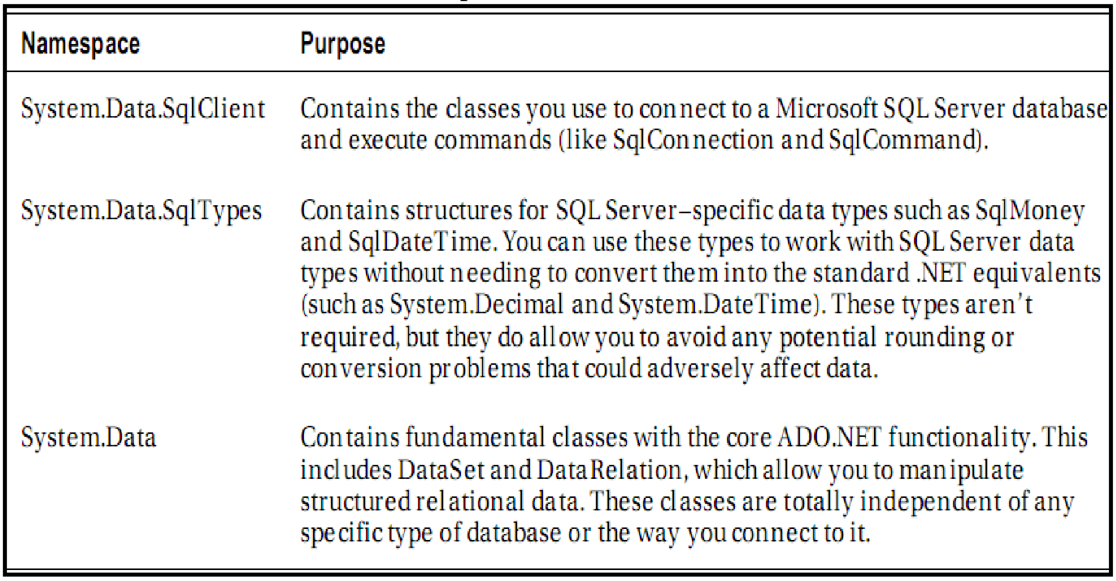
ADO.NET relies on the functionality in small set of core classes:

- To contain and manage data (such as DataSet, DataTable, DataRow, and DataRelation)
- To connect to a specific data source (such as Connection, Command, and DataReader)

DataSet: Its data container, customized for relational data.

Data providers are customized so that each one uses the best-performing way of interacting with its data source.

Ex: SQL Data Provider, Oracle data provider.



Namespace	Purpose
System.Data.SqlClient	Contains the classes you use to connect to a Microsoft SQL Server database and execute commands (like SqlConnection and SqlCommand).
System.Data.SqlTypes	Contains structures for SQL Server–specific data types such as SqlMoney and SqlDateTime. You can use these types to work with SQL Server data types without needing to convert them into the standard .NET equivalents (such as System.Decimal and System.DateTime). These types aren't required, but they do allow you to avoid any potential rounding or conversion problems that could adversely affect data.
System.Data	Contains fundamental classes with the core ADO.NET functionality. This includes DataSet and DataRelation, which allow you to manipulate structured relational data. These classes are totally independent of any specific type of database or the way you connect to it.

Figure 8.1 ADO.net namespaces for SQL Server Data Access

ADO.Net Namespaces

- Imports System.Data
- Imports System.Data.SqlClient

There are two types of Data Access.

1. Direct Data Access

- A most straightforward way to interact with a database.
- The copy of the information is not kept in memory. Instead, the information is used for a brief period while the database connection is open, and then close the connection as soon as possible.
- It is well suited to ASP.NET web pages, which do not need to keep a copy of their data in memory for long periods.
- To query information with simple data access, follow these steps:
    1. Create a Connection, Command, and DataReader objects.
    2. Use the DataReader to retrieve information from the database, and display it in control on a web form.
    3. Close the connection.
    4. Send the page to the user. At this point, the information which the user sees and the information in the database no longer have any connection, and all the ADO.NET objects have been destroyed.
- To add or update information, follow these steps:
    1. Create new Connection and Command objects.

    2. Execute the Command (with the appropriate SQL statement).

The sample for direct data access

```
 // Define the Select statement.
// Three pieces of information are needed: the unique id
// and the first and last name.
string selectSQL = "SELECT au_lname, au_fname, au_id FROM
Authors";

// Define the ADO.NET objects.

SqlConnection con = new SqlConnection(connectionString);
SqlCommand cmd = new SqlCommand(selectSQL, con);
SqlDataReader reader;
// Try to open database and read information.
try
 {
    con.Open();
    reader = cmd.ExecuteReader();
```

```
                    // For each item, add the author name to be displayed
                   // list box text, and store the unique ID in the Value property.
                    while (reader.Read())
                           {
                                  ListItem newItem = new ListItem();
                                 newItem.Text=reader["au_lname"]+","+reader["au_fname"];
                                 newItem.Value = reader["au_id"].ToString();
                                 lstAuthor.Items.Add(newItem);
                            }
                          reader.Close();
                           }
                    catch (Exception err)
                            {
                                      lblResults.Text = "Error reading list of names. ";
                                       lblResults.Text += err.Message;
                             }
                   finally
                           {
                                      con.Close();
                           }
                           }
```

2. Disconnected Data Access
   - A copy of the data is kept in the DataSet object so you can work with it after the database connection has been closed.
   - The user connects to the database just long enough to fetch your data and dump it into the DataSet, and then disconnects immediately.
   - The user fills the DataSet in much the same way that you connect a DataReader.
   - Datasets store a copy of data from the database tables.
   - Datasets cannot directly retrieve data from Databases.
   - DataAdapters are used to link Databases with DataSets.
   - If we see diagrammatically,

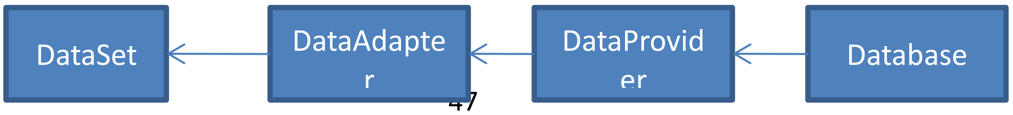


Figure 8.2 Disconnected Data Access

- The DataAdapter.Fill() method takes a DataSet and inserts one table of information.
- To access the individual DataRows, the user can loop through the Rows collection of the appropriate table. Each piece of information is accessed by using the field name, as it was with the DataReader.
- The sample showed in solved exercise.

Creating Connection to the data source

Before the user can retrieve or update data, he needs to make a connection to the data source. Write the database code inside a try/catch error-handling structure so the user can respond if an error does occur, and make sure to close the connection.

When creating a Connection object, the user needs to specify a value for its ConnectionString property. This ConnectionString defines all the information the computer needs to find the data source, log in, and choose an initial database.

- ***Data source:*** name of the server where the data source is located, “(localdb)\MSSQLlocalDB”
- ***Initial Catalog:*** Default database, change with Connection.ChangeDatabase( )
- ***Integrated security:*** Windows user account, SSPI (Security Support Provider Interface)
- ***ConnectionTimeout:*** How long the code will wait, in seconds, before generating an error if it cannot establish a database connection.

```
Eg:   SqlConnection myConnection = new SqlConnection();
      myConnection.ConnectionString = "Data Source=(localdb)\MSSQLLocalDB;" +
"Initial Catalog=Pubs;Integrated  Security=SSPI";
```

- All the database code in the application will use the same connection string. Therefore, it usually makes the most sense to store a connection string in a class member variable or, even better, a configuration file.

The <connectionStrings> section of the web.config file is a handy place to store the connection strings. Here’s an example:

```
        <configuration>
```

```
<connectionStrings>
<add name="Pubs" connectionString=
"DataSource=(localdb)\MSSQLLocalDB;Initial Catalog=Pubs;Integrated
Security=SSPI"/>
</connectionStrings>
. . . </configuration>
```

The user can then retrieve the connection string by name. First import the System.Web.Configuration namespace. Then the following code can be used:

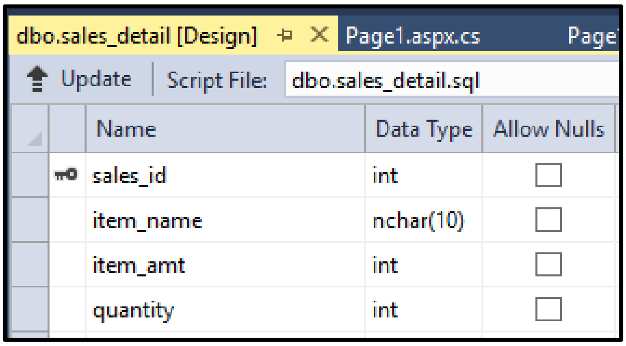
```
string connectionString =
WebConfigurationManager.ConnectionStrings["Pubs"].ConnectionString;
```

This approach helps ensure that all your web pages are using the same connection string.

II. SOLVED EXERCISE: Database- Supermarket Table – sales_detail

The web page contains a GridView and a DropDownList. The DropDownList is populated with sales_id using direct data access and based on the selected sales_id the corresponding details are retrieved from the database supermarket using disconnected data access and is populated in the GridView.

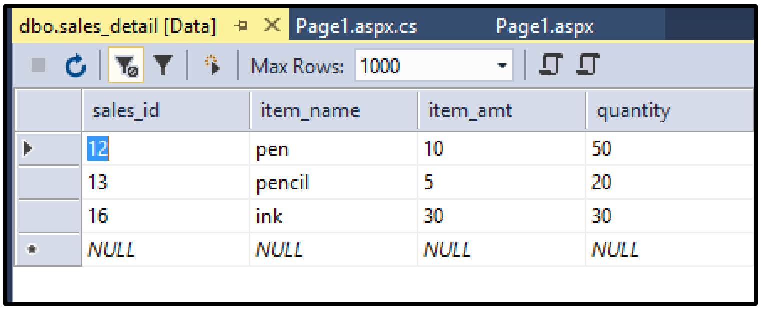
Table definition



Name	Data Type	Allow Nulls
sales_id	int	☐
item_name	nchar(10)	☐
item_amt	int	☐
quantity	int	☐

Figure 8.3 Table Definition

Table data



	sales_id	item_name	item_amt	quantity
▶	12	pen	10	50
	13	pencil	5	20
	16	ink	30	30
*	NULL	NULL	NULL	NULL

Figure 8.4 Table Data

Page1.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Page1.aspx.cs"
Inherits="Page1" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
<asp:Label runat="server" Text="Select the sales_id whose details you want to
view:" />  <br /><br />
<asp:DropDownList ID="ddl" runat="server" AutoPostBack="true"
OnSelectedIndexChanged="ddl_SelectedIndexChanged"/>
    <br /><br />
<asp:GridView ID="grid" runat="server" AutoGenerateColumns="false"
CellPadding="5" CellSpacing="5" BorderColor="black" BorderWidth="3"
AllowSorting="true" >
 <Columns>
```

```
<asp:BoundField DataField="item_name" HeaderText="Name"
SortExpression="item_name" HeaderStyle-BackColor="Gray"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="White"/>
<asp:TemplateField HeaderText = "Details" HeaderStyle-BackColor="Gray"
HeaderStyle-Font-Bold="true" HeaderStyle-ForeColor="White">
                    <ItemTemplate>
                        <b > Amount:</b>
                        <%# Eval("item_amt") %>
                        <br />
                        <b > Quantity:</b>
                        <%# Eval("quantity") %>
                        <br />
                    </ItemTemplate>
                </asp:TemplateField>
            </Columns>
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

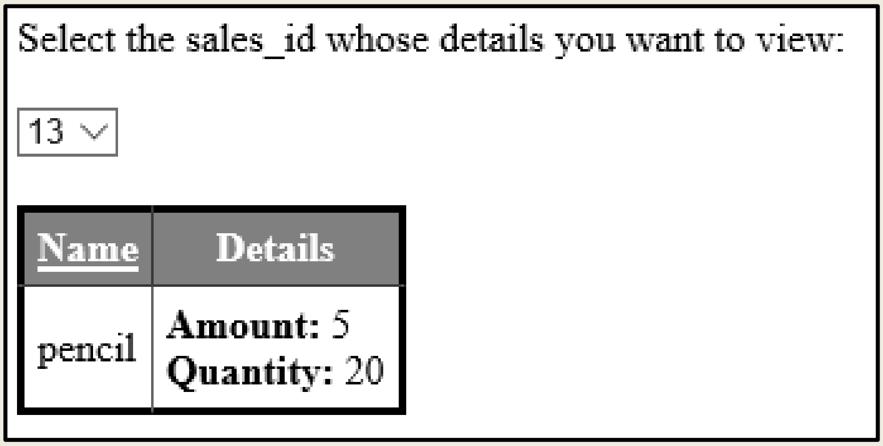
<u>Page1.aspx.cs</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;

public partial class Page1 : System.Web.UI.Page
{
    DataSet ds = new DataSet();
    protected void Page_Load(object sender, EventArgs e)
    {
        if(!IsPostBack)
//populate sales id drop down list
```

```
    {
       SqlConnection con = new SqlConnection();
       con.ConnectionString = @"Data Source=(localdb)\MSSQLLocalDB;Initial
       Catalog=Supermarket;Integrated Security=True;Pooling=False";
       try
          {
             con.Open();
             SqlCommand command = new SqlCommand("SELECT DISTINCT
             sales_id from sales_detail", con);
             SqlDataReader reader;
             reader = command.ExecuteReader();
             while (reader.Read())
                   {
                         ddl.Items.Add(reader["sales_id"].ToString());
                   }
            }
       catch (Exception ex)
             {        }
        finally
             {
                   con.Close();
             }
       }   }


protected void ddl_SelectedIndexChanged(object sender, EventArgs e)
  {
     SqlConnection con = new SqlConnection();
     con.ConnectionString = @"Data Source= (localdb)\MSSQLLocalDB;Initial
     Catalog=Supermarket; Integrated Security=True; Pooling=False";
     SqlCommand command = new SqlCommand("SELECT item_name, item_amt,
     quantity FROM sales_detail where sales_id = @sales_id", con);
     command.Parameters.AddWithValue("@sales_id", ddl.SelectedItem.Text);
     SqlDataAdapter adapter = new SqlDataAdapter(command);

      // All the information in transferred with one command.
      // This command creates a new DataTable (named sales_details)
      // inside the DataSet.
     adapter.Fill(ds, "sales_details");
```

```
        grid.DataSource = ds;
        grid.DataBind();
    }
  }
```

Output



Select the sales_id whose details you want to view:

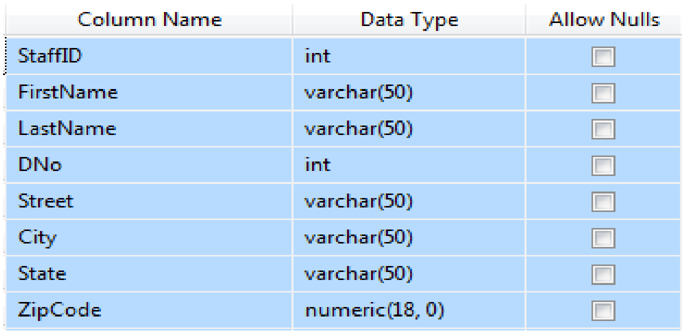
13

Name	Details
pencil	**Amount:** 5 **Quantity:** 20

Figure 8.5

III. LAB EXERCISES:

1) Consider a "**HouseKeeping**" database, containing "**Staff**" table whose schema is as shown below.

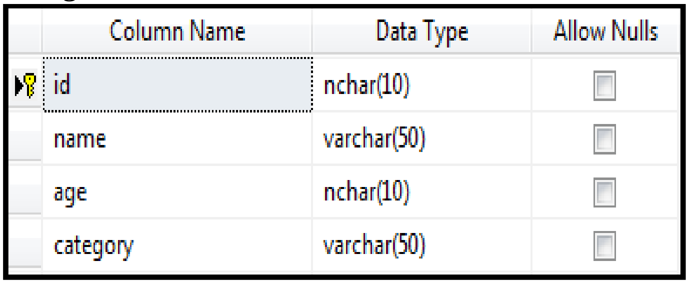


Column Name	Data Type	Allow Nulls
StaffID	int	☐
FirstName	varchar(50)	☐
LastName	varchar(50)	☐
DNo	int	☐
Street	varchar(50)	☐
City	varchar(50)	☐
State	varchar(50)	☐
ZipCode	numeric(18, 0)	☐

Figure 8.6

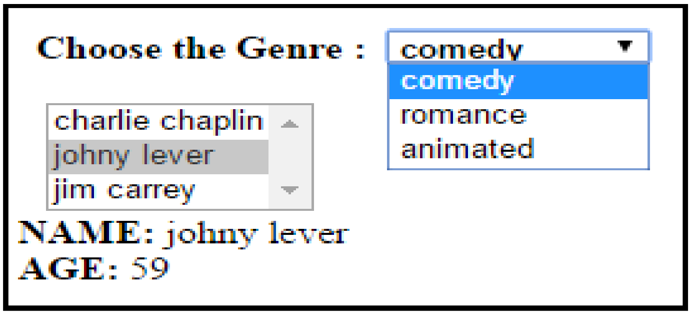
Create a web page consisting a DropDownList (displaying the StaffID), Label displaying all the details of the selected staff from the DropDownList, ListBox with few cities as items and “Update” button which updates the “City” in the database with the selected city from ListBox. Use disconnected data access.

2) Develop a web application that contains a DropDownList, a ListBox and a textarea as shown in figure 8.8 below. Use a dictionary collection with values “comedy”, “romance” and “animated” to bind with the DropDownList. On selecting the values from the DropDownList, corresponding names are to be retrieved from the table “Legends” in the database “Test” using direct data access and those are to be bound with the ListBox. The schema of the table “Legends” from “Test” is shown in figure 8.7. The textarea should display the name and age of the ListBox selection as shown in figure 8.8. Use appropriate data source controls for binding the textarea.



Column Name	Data Type	Allow Nulls
id	nchar(10)	☐
name	varchar(50)	☐
age	nchar(10)	☐
category	varchar(50)	☐

Figure 8.7



Choose the Genre : comedy

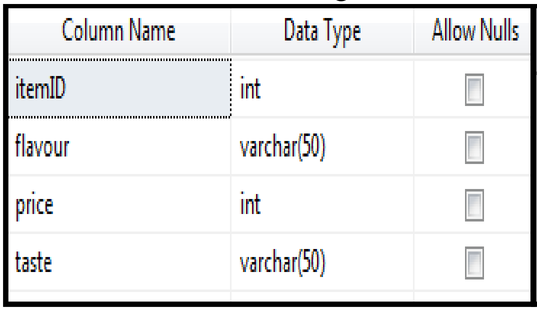
comedy
romance
animated

charlie chaplin
johny lever
jim carrey

NAME: johny lever
AGE: 59

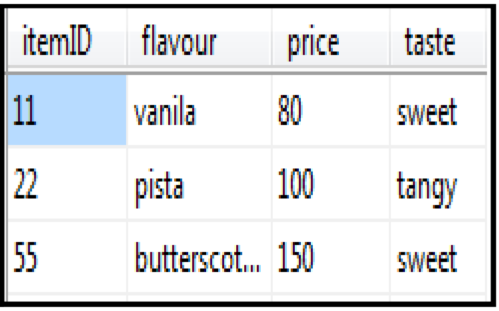
Figure 8.8

3) Consider the schema in figure 8.9 for table “Items” in the database “Products.”



Column Name	Data Type	Allow Nulls
itemID	int	☐
flavour	varchar(50)	☐
price	int	☐
taste	varchar(50)	☐

Figure 8.9



itemID	flavour	price	taste
11	vanila	80	sweet
22	pista	100	tangy
55	butterscot...	150	sweet

Figure 8.10

Modifying the price of “Vanilla” (data are shown in figure 8.10) to the value taken from a TextBox and update the database. Use direct data access and parameterized commands.

4) Consider the following tables:
WORKS(person-name,Company-name,Salary)
LIVES(Person_name, Street, City)
Assume Table data suitably. Design an ASP.NET webpage and include an option to insert data into WORKS table by accepting data from the user using TextBoxes. Also, include an option to retrieve the names of people who work for a particular company along with the cities they live in (particular company name must be accepted from the user). Use the Direct Data Access method to retrieve values from the Database.

IV. ADDITIONAL EXERCISES:

1. Assume a table “Institutes” with institute_id, name, and no_of_courses are the fields. Create a web page that retrieves all the data from “Institutes” table using disconnected data access method and displays only Institute names in the list box.

2. Create a web page with DropDownList, Textboxes and Buttons. Assume the table ‘Human’ with First name, Last name, Phone, Address and City as fields. When the page is loaded, only first names will be displayed in the drop-down list. On selecting the name, other details will be displayed in the respective TextBoxes. On clicking the update button, the table will be updated with new entries made in the text box. On clicking the delete button, the selected record will be deleted from the table, and the DropDownList is refreshed. Use direct access method and avoid

SQL injection attack.

LAB NO.: 7 **Date:**

WORKING WITH DATA - II

Objectives:

In this lab, the student will be able to:

- Learn to bind any server control to simple properties, collections or methods.
- Understand how to use SqlDataSource to work with Microsoft SQL Server.

I. DESCRIPTION

Repeated-value data binding works with the ASP.NET list controls. To use repeated-value binding, one of the below controls is linked to a data source (such as a field in a data table).

When DataBind( ) method is called, the control automatically creates a full list using all the corresponding values.

Some of the list controllers are:

- ListBox, DropDownList, CheckBoxList, and RadioButtonList
- HtmlSelect (Select)
- GridView

<u>Multiple Binding Sample:</u>

Step 1: Create and fill the collection

```
List<string> CSE = new List<string> ( );
CSE.Add("Operating Systems");
CSE.Add("DBS");
CSE.Add("FLAT");
CSE.Add("Compiler Design");
CSE.Add("Computer Networks");
```

Step 2: Define the binding for list controllers

```
MyListBox.DataSource = CSE;
MyCheckBoxList.DataSource = CSE;
MyRadioButtonList.DataSource = CSE;
```

Step 3: Activate Binding

```
this.DataBind( );
```

Data Binding with Dictionary Collection:

A dictionary collection is a special kind of collection in which every item is indexed with a specific key.

There are two basic dictionary-style collections in .NET:

- The Hashtable collection (in the System.Collections namespace)
- The Dictionary collection (in the System.Collections.Generic namespace)

Sample code:

Step 1: Create and fill the collection

```
Dictionary<int, string>CSE = new Dictionary<int, string>( );
CSE.Add(1,"Operating Systems");
CSE.Add(2,"DBS");
CSE.Add(3,"FLAT");
CSE.Add(4,"Compiler Design");
CSE.Add(5,"Computer Networks");
```

Step 2: Define the binding for list controllers

```
MyListBox.DataSource = CSE;
MyListBox.DataTextField = "Value";
MyListBox.DataValueField = "Key";
```

Step 3: Activate Binding

```
this.DataBind( );
```

Data Source Controls:

Data source controls allow the user to create data-bound pages without writing any data access code at all. They can retrieve data from a data source and supply it to bound controls. They can update the data source when edits take place. (GridView)

Some of the data source controls included in the .NET Framework are:

- SqlDataSource: This data source allows the user to connect to any data source that has an ADO.NET data provider.
- AccessDataSource: This data source allows the user to read and write the data in an Access database file (.mdb).
- ObjectDataSource: This data source allows the user to connect to a custom data access class.
- XmlDataSource: This data source allows the user to connect to an XML file.

The SqlDataSource:

```
<asp:SqlDataSource ID="SqlDS1" runat="server"
  ProviderName="System.Data.SqlClient"
 ConnectionString="<%$ ConnectionStrings:Student %>"
 SelectCommand="SELECT * FROM Student">
</asp:SqlDataSource>
```

The ProviderName property gives the name of the data provider factory that has the responsibility of creating the provider specific objects that the SqlDataSource needs to access the data source. The ConnectionString property points to the connection string defined in the web.config file.

The SqlDataSource supports four more properties:

- SelectCommand
- InsertCommand
- UpdateCommand
- DeleteCommand

Parameterized Commands:

The <SelectParameters> section inside the SqlDataSource tag defines each parameter that are referenced by the SelectCommand and tells the SqlDataSource where to find the value it should use. The parameter has to be mapped to a value in control. (Example is given in Sample Exercise)

QueryString Parameter:

When the user wants to map the parameter to a value in the QueryString instead of any other control, this parameter can be used.

Here is a Button event handler code in the first page to copy the selected product to the query string and redirect the page.

```
protected void cmdGo_Click(object sender, EventArgs e)
{
      if (lstProduct.SelectedIndex != -1)
      {
              Response.Redirect("QueryParameter2.aspx?prodID=" +
              lstProduct.SelectedValue);
      }
}
```

Finally, the second page can bind the GridView according to the ProductID value that’s supplied in the Query String:

```
<asp:SqlDataSource ID="sourceProductDetails" runat="server"
ProviderName="System.Data.SqlClient"
ConnectionString="<%$ ConnectionStrings:Northwind %>"
SelectCommand="SELECT * FROM Products WHERE ProductID=@ProductID">
<SelectParameters>
    <asp:QueryStringParameter Name="ProductID" QueryStringField="prodID" />
</SelectParameters>
</asp:SqlDataSource>
<asp:GridView ID="GridProduct" runat="server"
DataSourceID="sourceProductDetails" />
```

GridView:

The GridView is an extremely flexible grid control that displays a multicolumn table. Each record in your data source becomes a separate row in the grid. Each field in the record becomes a separate column in the grid. The GridView provides a DataSource property for the data object the user wants to display.

How to use XmlDataSource to fetch data from xml file?

Step 1: Create an xml file

1. In Solution Explorer, right-click the App_Data folder, and then click AddNew Item.
2. Under Visual Studio installed templates, click XML file.
3. In the Name box, type Musicstore.xml. Click Add.
4. A new .xml file is created containing only the XML directive.
5. Fill it with required content and save the file.

Step 2: Displaying XML data in GridView control

1. Open the Default.aspx file, and then switch to Design view.
2. In the Toolbox, from the Data group, drag an XmlDataSource control onto the page.
3. On the XmlDataSourceTasks menu, click Configure Data Source.
4. The Configure Data Source <DataSourceName> dialog box appears.
5. In the Data file box, type ~/App_Data/Musicstore.xml. Click OK.
6. In the Toolbox, from the Data group, drag a GridView control onto the page.
7. On the GridView Tasks menu, in the Choose Data Source list,
   click XmlDataSource1.

II. SOLVED EXERCISE:

Design a website to load the names of students from a ‘Student’ table in a database to a DropDownList. On selecting the student name from the list, the entire information of that student must be displayed in a GridView. Use data source control for fetching data.

Default.aspx.cs contents:

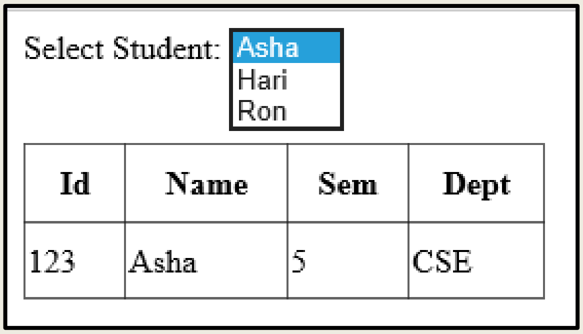
```
<form id="form1" runat="server">

<asp:SqlDataSource ID="SqlDS1" runat="server"
ConnectionString="<%$ConnectionStrings:Student%>"
SelectCommand="SELECT * FROM Student">
</asp:SqlDataSource>

Select Student:
<asp:DropDownList ID="DropDownList1" runat="server" DataSourceID="SqlDS1"
DataTextField="Name" DataValueField="id" AutoPostBack="True">
</asp:DropDownList>
 <br /> <br /> <br />

<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDS2"
AutoGenerateEditButton="False" Height="76px" Width="254px" >
</asp:GridView>

<asp:SqlDataSource ID="SqlDS2"  ConnectionString="<%$
ConnectionStrings:Student %>" runat="server"
SelectCommand="SELECT * FROM Student WHERE id=@id" >
<SelectParameters>
<asp:ControlParameter Name="id" ControlID="DropDownList1"
PropertyName="SelectedValue" />
</SelectParameters>
</asp:SqlDataSource>
</form>
```

Web.config:

```
<connectionStrings>
```

```
    <add name="Student" connectionString="Data
Source=(localdb)\MSSQLlocalDB;Initial  Catalog=Test;Integrated Security=True"/>
</connectionStrings>
```

Output:



Select Student: Asha / Hari / Ron

Id	Name	Sem	Dept
123	Asha	5	CSE

Figure 9.1

III. LAB EXERCISES:

1) Develop a website for a juice shop wherein the user is asked to choose from a list of fruits (use checkboxlist) and a list of ice creams (use radiobuttonlist) for the milkshake to be rendered. Provide a button that needs to display both the selections in a label along with price (random) for the milkshake. Use data binding with a list to populate the fruit and ice cream lists.

2) Develop an ASP.NET website to display the names of the tour packages available with a travel agency in a Listbox using Sqldatasource in Page1. Use Sqldatasource and GridView control to display the entire detail of the package in Page2 based on the selection made in Page1 (the selected value in Page1 has to be sent to Page2 via querystring and in Page2 use Querystring parameter in sqldatasource control). Sample table data given below.

Place	Days	Cost
Kulu-Manali	5	20000
Shimla-Darjeeling	6	15000
Ooty-Kodaikanal	4	20000

3) Consider a table named City whose fields are as follows.

City:
Id, StateID, CityName
Develop a web application which allows a user to choose a State from the first DropDownList (bind it with a dictionary collection of at least five states during page load). Based on the value selected in the first DropDownList, the second DropDownList should be populated with the cities in that particular State fetched from the City table (use data source control). The keys of the dictionary can be mapped to the values of StateID in the table. The dictionary key has to be set as a DropDownList index, and the dictionary value has to be set as display value for the DropDownList.

4) Develop a web application that contains a DropDownList, a ListBox and a GridView. Use a list with values comedy, romance and animated (various categories) to bind with the DropDownList. The ListBox should display the names of the actors who belong to the category as that of the genre selected in the DropDownList. The GridView must display entire details of the actor based on the name selected in the ListBox. Use data source control for all database operations. The schema of the 'Actors' table is given below.(id can be set as a primary key)

Column name	Data type
id	nchar(10)
name	varchar(10)
age	nchar(10)
category	varchar(10)

IV. ADDITIONAL EXERCISES:

1) Create a website to display the details and user reviews on various books available in a bookstore stored in an XML file using a GridView and an XmlDataSource control.

   Sample xml file content:

```
<?xml version="1.0" standalone="yes"?>
<bookstore>
    <book ISBN="10-000000-001"
        title="The Iliad and The Odyssey"
        price="12.95">
    <comments>
        <userComment rating="4"
            comment="Best translation I've read." />
        <userComment rating="2"
            comment="I like other versions better." />
      </comments>
   </book>
   <book ISBN="11-000000-002"
        title="Computer Dictionary"
        price="24.95" >
      <comments>
         <userComment rating="3"
             comment="A valuable resource." />
      </comments>
   </book>
</bookstore>
```

2) Develop a website that displays a list of courses (CheckBoxList) offered by the CSE dept as open electives. Use data binding using the list to show the courses. When anyone course has been selected the user should be navigated to a new page where he/she can read the contents of the course along with prerequisites, faculty handling the subject and general review on the course. The data of every such course must be stored in the database and retrieved using data source control.

LAB NO.: 8 **Date:**

GRIDVIEW, FILES & XML

Objectives:

In this lab, student will be able to:

- Understand and use the GridView control to efficiently display data from database in Web pages.
- Learn about support for files in ASP.NET and to implement file operations in Web pages.
- Create XML files and to perform read/write operations on those files using .NET concepts.

I. DESCRIPTION

The GridView is an extremely flexible grid control that displays a multicolumn table. Each record in your data source becomes a separate row in the grid. Each field in the record becomes a separate column in the grid. The functionality of the GridView includes features for automatic paging, sorting, selecting, and editing.

How to display table data in a GridView?
First define a SqlDataSource to perform a query as follows:

```
<asp:SqlDataSource ID = "sourceProducts" runat = "server"
ConnectionString = " <%$ ConnectionStrings:Northwind %> "
SelectCommand = "SELECT ProductID, ProductName, UnitPrice FROM Products" />
```

Next, set the GridView.DataSourceID property to link the data source to the grid:

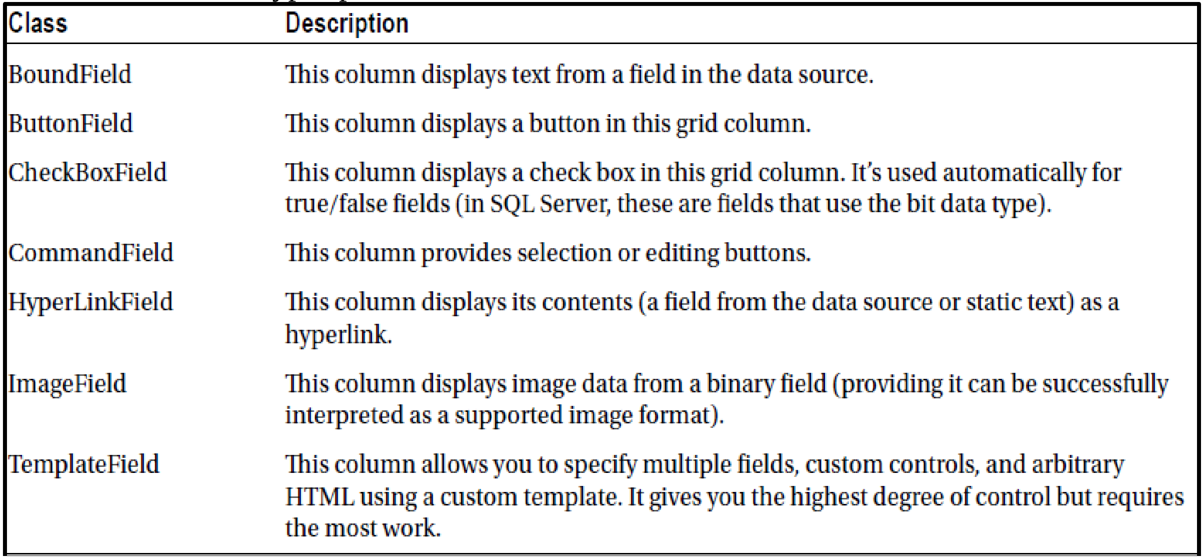
```
<asp:GridView ID = "GridView1" runat = "server" DataSourceID = "sourceProducts"/>
```

How to define columns in the GridView as per user's choice?
Set the GridView.AutoGenerateColumns property to false and use code similar to the one shown below.

```
<asp:GridView ID = "GridView1" runat = "server" DataSourceID = "sourceProducts"
AutoGenerateColumns = "False">
<Columns>
    <asp:BoundField DataField = "ProductID" HeaderText = "ID" />
    <asp:BoundField DataField = "ProductName" HeaderText = "Product Name" />
    <asp:BoundField DataField = "UnitPrice" HeaderText = "Price" />
```

```
</Columns>
</asp:GridView>
```

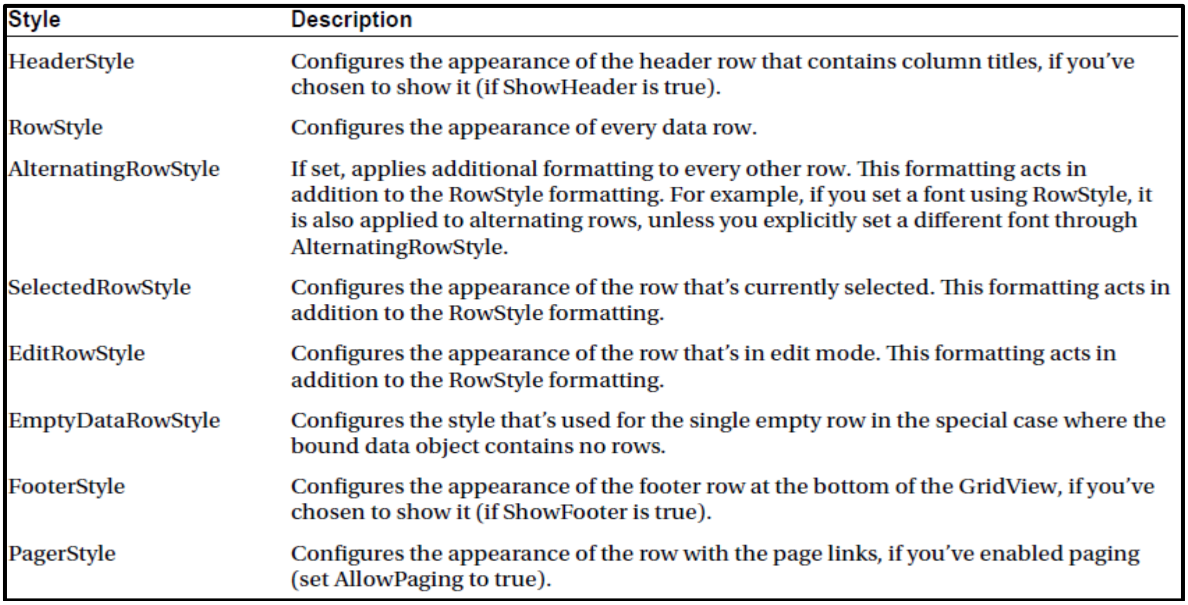
The several column types possible with a GridView are shown in the table below.



Class	Description
BoundField	This column displays text from a field in the data source.
ButtonField	This column displays a button in this grid column.
CheckBoxField	This column displays a check box in this grid column. It's used automatically for true/false fields (in SQL Server, these are fields that use the bit data type).
CommandField	This column provides selection or editing buttons.
HyperLinkField	This column displays its contents (a field from the data source or static text) as a hyperlink.
ImageField	This column displays image data from a binary field (providing it can be successfully interpreted as a supported image format).
TemplateField	This column allows you to specify multiple fields, custom controls, and arbitrary HTML using a custom template. It gives you the highest degree of control but requires the most work.

Figure 10.1

GridView Styles:



Style	Description
HeaderStyle	Configures the appearance of the header row that contains column titles, if you've chosen to show it (if ShowHeader is true).
RowStyle	Configures the appearance of every data row.
AlternatingRowStyle	If set, applies additional formatting to every other row. This formatting acts in addition to the RowStyle formatting. For example, if you set a font using RowStyle, it is also applied to alternating rows, unless you explicitly set a different font through AlternatingRowStyle.
SelectedRowStyle	Configures the appearance of the row that's currently selected. This formatting acts in addition to the RowStyle formatting.
EditRowStyle	Configures the appearance of the row that's in edit mode. This formatting acts in addition to the RowStyle formatting.
EmptyDataRowStyle	Configures the style that's used for the single empty row in the special case where the bound data object contains no rows.
FooterStyle	Configures the appearance of the footer row at the bottom of the GridView, if you've chosen to show it (if ShowFooter is true).
PagerStyle	Configures the appearance of the row with the page links, if you've enabled paging (set AllowPaging to true).

Figure 10.2

How to change formatting for a specific row or just highlight a single cell?

The GridView.RowDataBound event can be used for the above purpose. This event is raised for each row, just after it is filled with data. At this point, the current row can be accessed as a GridViewRow object. The GridViewRow.DataItem property provides the data object for the given row and the GridViewRow.Cells collection allows retrieving the row content. The GridViewRow can be used to change colors and alignment, add or remove child controls, and so on.

The following example handles the RowDataBound event and changes the background color to highlight high prices (those more expensive than $50):

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
        if (e.Row.RowType == DataControlRowType.DataRow)
        {
                // Get the price for this row.
                decimal price = (decimal)DataBinder.Eval(e.Row.DataItem,"UnitPrice");
                if (price > 50)
                {
                        e.Row.BackColor = System.Drawing.Color.Maroon;
                        e.Row.ForeColor = System.Drawing.Color.White;
                        e.Row.Font.Bold = true;
                }
        }
}
```

Editing with the GridView:

To allow editing of a row in GridView, a CommandField column has to be added and ShowEditButton property has to be set to true.

```
<asp:GridView ID = "gridProducts" runat = "server" DataSourceID = "sourceProducts"
AutoGenerateColumns = "False" DataKeyNames = "ProductID">
<Columns>
<asp:BoundField DataField = "ProductID" HeaderText = "ID" ReadOnly = "True" />
<asp:BoundField DataField = "ProductName" HeaderText = "Product Name"/>
<asp:BoundField DataField = "UnitPrice" HeaderText = "Price" />
<asp:CommandField  ShowEditButton = "True" />
</Columns>
</asp:GridView>
```

Sorting:

The GridView sorting features allow the user to reorder the results in the GridView by clicking a column header. To enable sorting in the GridView.AllowSorting property must be set to true. Next, a SortExpression for each column that can be sorted has to be defined.

```
<asp:BoundField  DataField = "ProductName" HeaderText = "Product Name"
SortExpression = "ProductName" />
```

Paging:

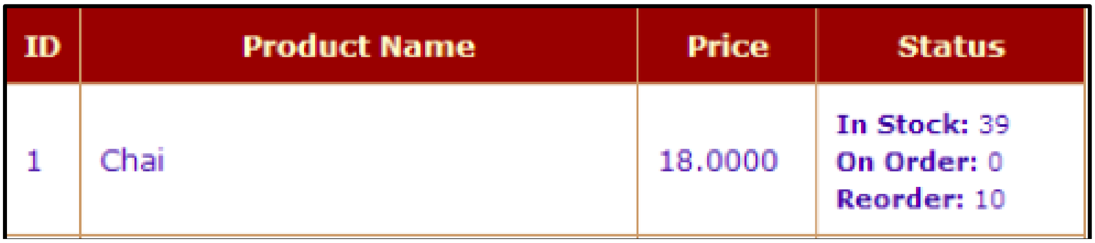
To use automatic paging, the GridView.AllowPaging property has to be set to true and the PageSize has to be set to determine how many rows are allowed on each page.

```
<asp:GridView ID = "GridView1" runat = "server" DataSourceID = "sourceProducts"
PageSize = "10" AllowPaging = "True" > </asp:GridView>
```

Using GridView Templates:

If multiple values have to be placed in the same cell then a TemplateField has to be used. Suppose if the three fields of the table namely in-stock, on-order and reorder level need to be combined, then the following code can be used.

```
<asp:TemplateField HeaderText = "Status">
        <ItemTemplate>
                <b > In Stock:</b>
                <%# Eval("UnitsInStock") % > <br />
                <b > On Order:</b>
                <%# Eval("UnitsOnOrder") % > <br />
                <b > Reorder:</b>
                <%# Eval("ReorderLevel") %>
        </ItemTemplate>
</asp:TemplateField>
```



ID	Product Name	Price	Status
1	Chai	18.0000	**In Stock:** 39 **On Order:** 0 **Reorder:** 10

Figure 10.3

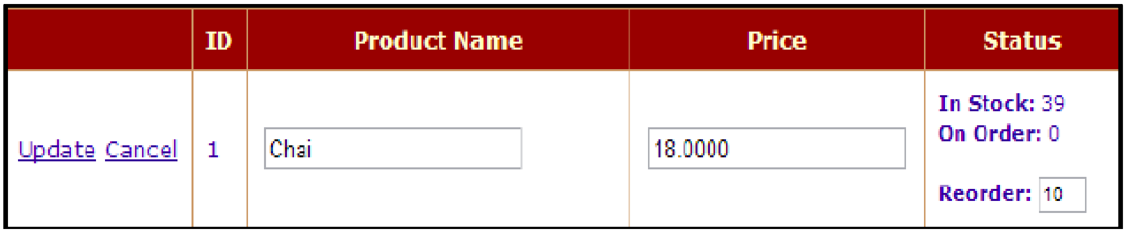
The Column ‘Status’ in the above snapshot is generated using the defined TemplateField.

The template only has access to the fields that are in the bound data object. So if the UnitsInStock, UnitsOnOrder, and ReorderLevel fields have to be shown, the SqlDataSource query has to be designed to return this information.

Editing with Templates:

Suppose it is required to edit any item in the TemplateField then an EditItemTemplate section has to be added as shown below.

```
<asp:TemplateField HeaderText = "Status">
        <ItemTemplate>
                <b > In Stock:</b > <%# Eval("UnitsInStock") % > <br />
                <b > On Order:</b > <%# Eval("UnitsOnOrder") % > <br />
                <b > Reorder:</b > <%# Eval("ReorderLevel") %>
        </ItemTemplate>
        <EditItemTemplate>
        <b > In Stock:</b > <%# Eval("UnitsInStock") % > <br />
        <b > On Order:</b > <%# Eval("UnitsOnOrder") % > <br /> < br />
        <b > Reorder:</b>
        <asp:TextBox Text = ' < %# Bind("ReorderLevel") % > ' Width = "25px"
        runat = "server" id = "txtReorder" />
        </EditItemTemplate>
</asp:TemplateField>
```



	ID	Product Name	Price	Status
Update Cancel	1	Chai	18.0000	In Stock: 39 On Order: 0 Reorder: 10

Figure 10.4

Files:

The simplest level of file access involves just retrieving information about existing files and directories and performing typical file system operations such as copying files and creating directories.

.NET provides five basic classes for retrieving this sort of information. They are all located in the System.IO namespace (and, incidentally, can be used in desktop applications in the same way they are used in web applications).

They include the following:

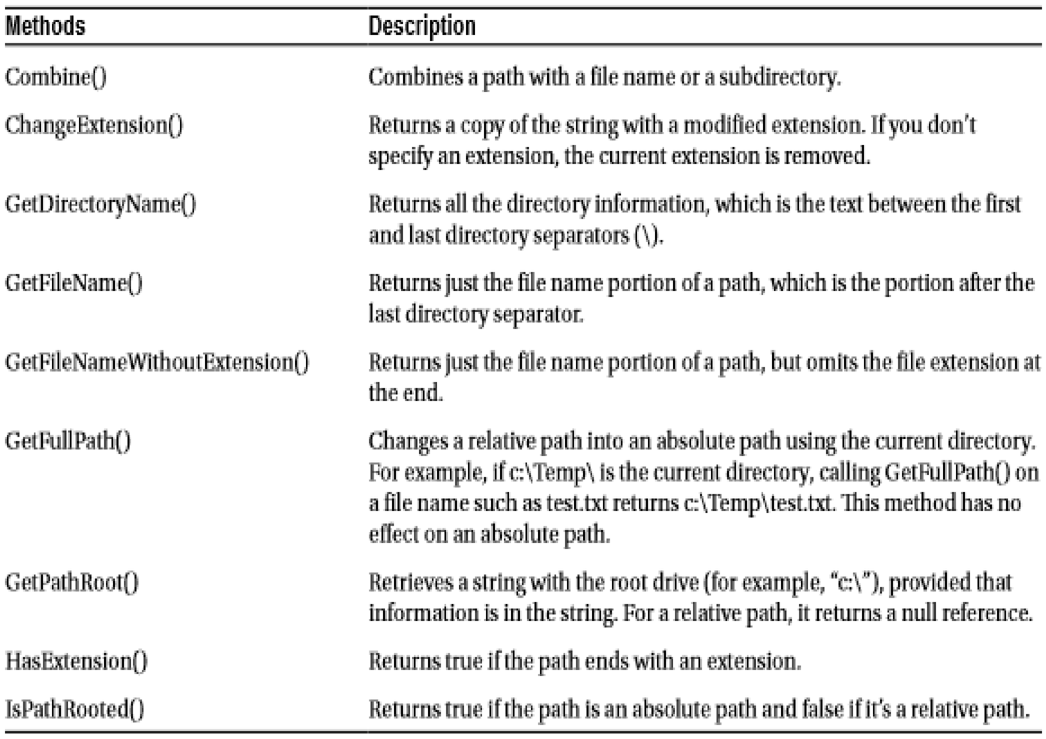
The Directory and File classes, which provide static methods that allow the user to retrieve information about any files and directories visible from the server.

The DirectoryInfo and FileInfo classes, which use similar instance methods and properties to retrieve the same sort of information as Directory and File classes.

The DriveInfo class, which provides static methods that allow the user to retrieve information about a drive and the amount of free space it provides.

.NET also includes a helper class named Path in the same System.IO namespace. The Path class does not include any real file management functionality. It simply provides a few static methods that are useful when manipulating strings that contain file and directory paths.

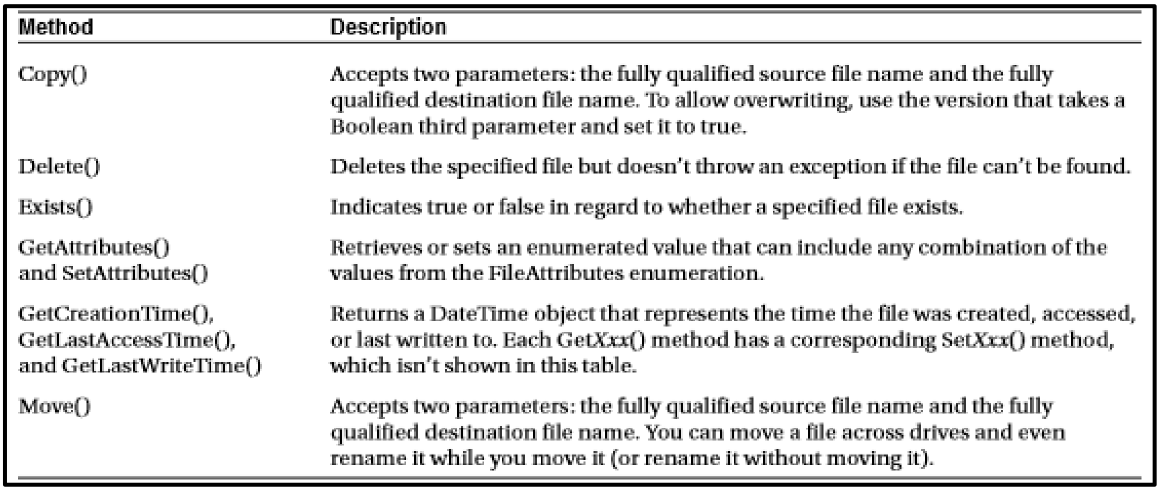
The following are the various methods under the Path class.



Methods	Description
Combine()	Combines a path with a file name or a subdirectory.
ChangeExtension()	Returns a copy of the string with a modified extension. If you don't specify an extension, the current extension is removed.
GetDirectoryName()	Returns all the directory information, which is the text between the first and last directory separators (\).
GetFileName()	Returns just the file name portion of a path, which is the portion after the last directory separator.
GetFileNameWithoutExtension()	Returns just the file name portion of a path, but omits the file extension at the end.
GetFullPath()	Changes a relative path into an absolute path using the current directory. For example, if c:\Temp\ is the current directory, calling GetFullPath() on a file name such as test.txt returns c:\Temp\test.txt. This method has no effect on an absolute path.
GetPathRoot()	Retrieves a string with the root drive (for example, "c:\"), provided that information is in the string. For a relative path, it returns a null reference.
HasExtension()	Returns true if the path ends with an extension.
IsPathRooted()	Returns true if the path is an absolute path and false if it's a relative path.

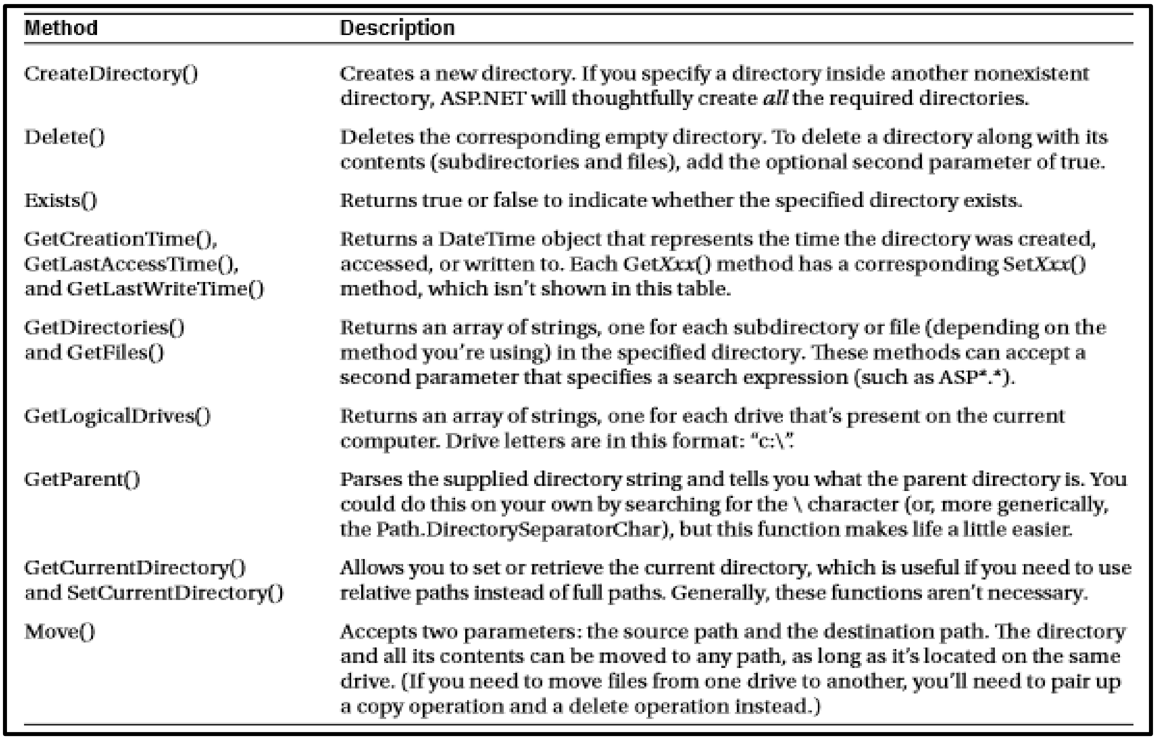
Figure 10.5 Path Class Methods

The Directory and File classes provide some useful static methods. Figure 10.6 and Figure 10.7 show an overview of the most important methods. Most of these methods take the same parameter: a fully qualified pathname identifying the directory or file for which the user wants the operation to act on.



Method	Description
Copy()	Accepts two parameters: the fully qualified source file name and the fully qualified destination file name. To allow overwriting, use the version that takes a Boolean third parameter and set it to true.
Delete()	Deletes the specified file but doesn't throw an exception if the file can't be found.
Exists()	Indicates true or false in regard to whether a specified file exists.
GetAttributes() and SetAttributes()	Retrieves or sets an enumerated value that can include any combination of the values from the FileAttributes enumeration.
GetCreationTime(), GetLastAccessTime(), and GetLastWriteTime()	Returns a DateTime object that represents the time the file was created, accessed, or last written to. Each Get*Xxx*() method has a corresponding Set*Xxx*() method, which isn't shown in this table.
Move()	Accepts two parameters: the fully qualified source file name and the fully qualified destination file name. You can move a file across drives and even rename it while you move it (or rename it without moving it).

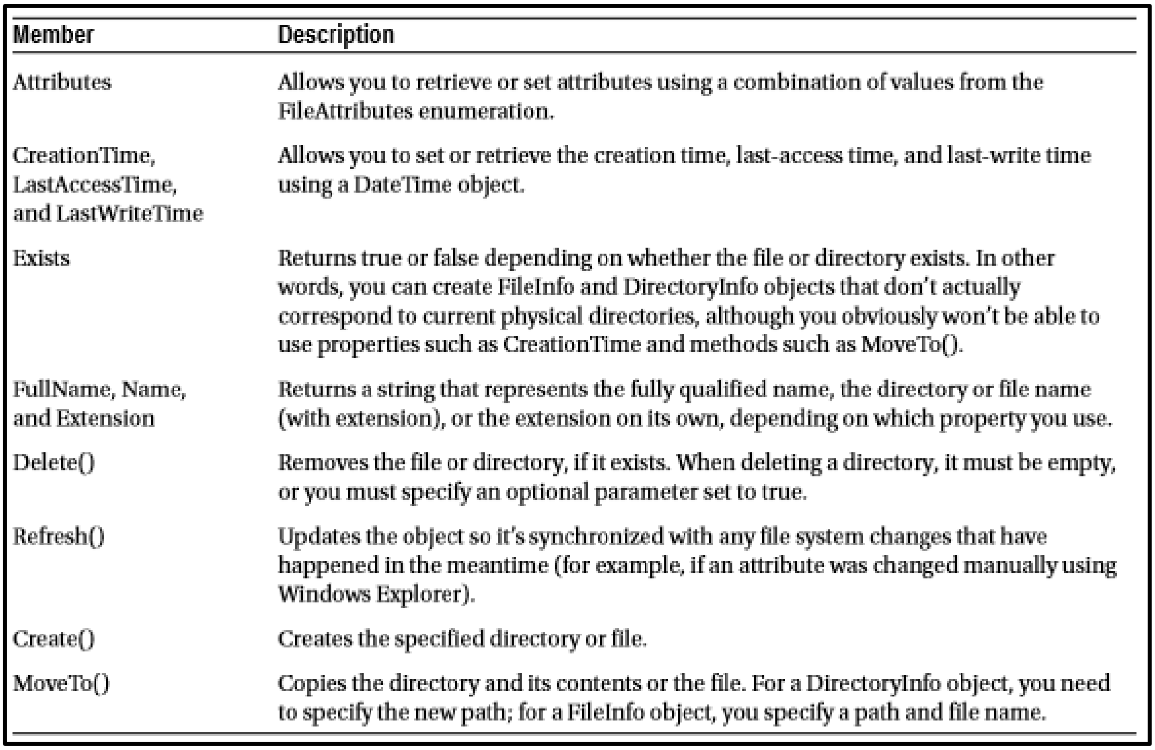
Figure 10.6 File class methods



Method	Description
CreateDirectory()	Creates a new directory. If you specify a directory inside another nonexistent directory, ASP.NET will thoughtfully create *all* the required directories.
Delete()	Deletes the corresponding empty directory. To delete a directory along with its contents (subdirectories and files), add the optional second parameter of true.
Exists()	Returns true or false to indicate whether the specified directory exists.
GetCreationTime(), GetLastAccessTime(), and GetLastWriteTime()	Returns a DateTime object that represents the time the directory was created, accessed, or written to. Each Get*Xxx*() method has a corresponding Set*Xxx*() method, which isn't shown in this table.
GetDirectories() and GetFiles()	Returns an array of strings, one for each subdirectory or file (depending on the method you're using) in the specified directory. These methods can accept a second parameter that specifies a search expression (such as ASP*.*).
GetLogicalDrives()	Returns an array of strings, one for each drive that's present on the current computer. Drive letters are in this format: "c:\".
GetParent()	Parses the supplied directory string and tells you what the parent directory is. You could do this on your own by searching for the \ character (or, more generically, the Path.DirectorySeparatorChar), but this function makes life a little easier.
GetCurrentDirectory() and SetCurrentDirectory()	Allows you to set or retrieve the current directory, which is useful if you need to use relative paths instead of full paths. Generally, these functions aren't necessary.
Move()	Accepts two parameters: the source path and the destination path. The directory and all its contents can be moved to any path, as long as it's located on the same drive. (If you need to move files from one drive to another, you'll need to pair up a copy operation and a delete operation instead.)

Figure 10.7 Directory Class Methods

The DirectoryInfo and FileInfo classes mirror the functionality in the Directory and File classes. Also, they make it easy to walk through the directory and file relationships. They share a common set of properties and methods because they derive from the common FileSystemInfo base class.



Member	Description
Attributes	Allows you to retrieve or set attributes using a combination of values from the FileAttributes enumeration.
CreationTime, LastAccessTime, and LastWriteTime	Allows you to set or retrieve the creation time, last-access time, and last-write time using a DateTime object.
Exists	Returns true or false depending on whether the file or directory exists. In other words, you can create FileInfo and DirectoryInfo objects that don't actually correspond to current physical directories, although you obviously won't be able to use properties such as CreationTime and methods such as MoveTo().
FullName, Name, and Extension	Returns a string that represents the fully qualified name, the directory or file name (with extension), or the extension on its own, depending on which property you use.
Delete()	Removes the file or directory, if it exists. When deleting a directory, it must be empty, or you must specify an optional parameter set to true.
Refresh()	Updates the object so it's synchronized with any file system changes that have happened in the meantime (for example, if an attribute was changed manually using Windows Explorer).
Create()	Creates the specified directory or file.
MoveTo()	Copies the directory and its contents or the file. For a DirectoryInfo object, you need to specify the new path; for a FileInfo object, you specify a path and file name.

Figure 10.8 DirectoryInfo and FileInfo class common methods

To create a DirectoryInfo or FileInfo object, the user should specify the full path in the constructor:

```
DirectoryInfo myDirectory = new DirectoryInfo(@"c:\Temp");

FileInfo myFile = new FileInfo(@"c:\Temp\readme.txt");
```

This path may or may not correspond to a real physical file or directory. If it does not, then the Create() method can be used to create the corresponding file or directory:

```
// Define the new directory and file.
```

```
DirectoryInfo myDirectory = new DirectoryInfo(@"c:\Temp\Test");

FileInfo myFile = new FileInfo(@"c:\Temp\Test\readme.txt");
```

// Now create them. Order here is important. A file cannot be created in a directory that doesn't exist yet.

```
myDirectory.Create();

myFile.Create();
```

The .NET Framework makes it easy to create simple “flat” files in text or binary format. The user can write to a text file and read from a text file using a StreamWriter and a StreamReader—dedicated classes that abstract away the process of file interaction.

```
// Define a StreamWriter (which is designed for writing text files).

StreamWriter w;  // Create the file, and get a StreamWriter for it.

w = File.CreateText(@"c:\Temp\myfile.txt");
```

Using the StreamWriter, the user can call the WriteLine() method to add information to the file. The WriteLine() method is overloaded so it can write many simple data types, including strings, integers, and other numbers. These values are essentially all converted into strings when they are written to a file and must be converted back manually into the appropriate types when the file has to be read.

```
w. WriteLine("This file generated by ASP.NET");      // Write a string.

w. WriteLine(42);                                   // Write a number.
```

On completing the operations with the file, the user must make sure to close it by calling the Close() or Dispose() method. Otherwise, the changes may not be properly written to disk and the file could be locked open.

```
w.Close();
```

To read the information, the user can use the corresponding StreamReader class. It provides a ReadLine() method that gets the next available value and returns it as a string. ReadLine() starts at the first line and advances the position to the end of the file, one line at a time.

```
StreamReader r = File.OpenText(@"c:\myfile.txt");

string inputString; inputString = r.ReadLine();    // = "This file generated by
ASP.NET" inputString = r.ReadLine();    // = "42"
```

ReadLine() returns a null reference when there is no more data in the file. This means that the user can read all the data in a file using code like this:

// Read and display the lines from the file until the end // of the file is reached.

```
string line;
do
{
     line = r.ReadLine();
     if (line != null)
                {
          // (Process the line here.)
      }
}
 while (line != null);
}
```

As when writing to a file, Close the file once the operations are done:

```
r.Close();
```

The user can also read and write to binary files. Binary data use space more efficiently but also create files that aren't human-readable. To open a file for binary writing, the user needs to create a new BinaryWriter object. The constructor accepts a stream, which can be retrieved using the File.OpenWrite() method. Here's the code to open the file c:\binaryfile. Bin for binary writing:

```
FileStream fs = File.OpenWrite(@"c:\binaryfile.bin");
BinaryWriter w = new BinaryWriter(fs);
```

.NET concentrates on stream objects, rather than on the source or destination for the data. This means that the user can write binary data to any type of stream, whether it represents a file or some other type of storage location, using the same code. Also, writing to a binary file is almost the same as writing to a text file.

```
string str = "ASP.NET Binary File Test";
int integer = 42;
w.Write(str);
w.Write(integer);
w.Close();
```

Reading data from a binary file is easy, but not quite as easy as reading data from a text file. The problem is that the user needs to know the type of data to retrieve. To retrieve a string, the ReadString() method must be used. To retrieve an integer ReadInt32() must be used.

```
BinaryReader r = new BinaryReader(File.OpenRead(@"c:\binaryfile.bin"));
string str; int integer;
str = r.ReadString();
 integer = r.ReadInt32();
r.Close();
```

XML

XML is designed as an all-purpose format for organizing data. It is an all-purpose way to identify any type of data using elements. These elements use the same sort of format found in an HTML file, but while HTML elements indicate formatting, XML elements indicate content.

```
<?xml version="1.0"?>
 <SuperProProductList>
 <Product>
<ID>1</ID>
<Name>Chair</Name>
<Price>49.33</Price>
<Available>True</Available>
<Status>3</Status>
```

```
</Product>
</SuperProProductList>
```

This format is understandable. Every product item is enclosed in a <Product> element, and every piece of information has its element with an appropriate name. Elements are nested several layers deep to show relationships. Essentially, XML provides the basic element syntax, and the programmer defines the elements which need to be used. That is why XML is often described as a metalanguage— it is a language used to create the developer's language. In the SuperProProductList example, this custom XML language defines elements such as <Product>, <ID>, <Name>, and so on.

.NET provides a rich set of classes for XML manipulation in several namespaces that start with System.Xml. One of the simplest ways to create or read any XML document is to use the basic XmlTextWriter and XmlTextReader classes. These classes work like their StreamWriter and StreamReader relatives, except that they write and read XML documents instead of ordinary text files. A sample is as shown below:

```
FileStream fs = new FileStream(file, FileMode.Create);
XmlTextWriter w = new XmlTextWriter(fs, null);
 w.WriteStartDocument();
w.WriteStartElement("SuperProProductList");
w.WriteComment("This file generated by the XmlTextWriter class.");
  // Write the first product.
 w.WriteStartElement("Product");
w.WriteAttributeString("ID", "1");
w.WriteAttributeString("Name", "Chair");
w.WriteStartElement("Price");
w.WriteString("49.33");
w.WriteEndElement();
w.WriteEndElement();
```

Reading the XML document in the code is just as easy with the corresponding XmlTextReader class. The XmlTextReader moves through the document from top to bottom, one node at a time. Call the Read() method to move to the next node. This method returns true if there are more nodes to read or false once it has read the final node. The current node is provided through the properties of the XmlTextReader class, such as

NodeType and Name. A node is a designation that includes comments, whitespace, opening tags, closing tags, content, and even the XML declaration at the top of the file. The code snippet is given below:

```
// Use a StringWriter to build up a string of HTML that // describes the
information read from the XML document.

 StringWriter writer = new StringWriter();
// Parse the file, and read each node.
while (r.Read())
 {    // Skip whitespace.
      if (r.NodeType == XmlNodeType.Whitespace)
            continue;
      writer.Write("<b>Type:</b> ");
      writer.Write(r.NodeType.ToString());
      writer.Write("<br>");
            // The name is available when reading the opening and closing tags
            // for an element. It's not available when reading the inner content.
     if (r.Name != "")
             {
                    writer.Write("<b>Name:</b> ");
                    writer.Write(r.Name);
                    writer.Write("<br>");
            }
            // The value is when reading the inner content.
    if (r.Value != "")
            {
                    writer.Write("<b>Value:</b> ");
                    writer.Write(r.Value);
                    writer.Write("<br>");
            }
    if (r.AttributeCount > 0)
            {
                    writer.Write("<b>Attributes:</b> ");
                    for (int i = 0; i < r.AttributeCount; i++)
            {
                              writer.Write("  ");
                              writer.Write(r.GetAttribute(i));
                              writer.Write("  ");
```

```
                }
                        writer.Write("<br>");
                }
                        writer.Write("<br>");
        }
```

XDocument class provides a different approach to XML data. It provides an in-memory model of an entire XML document. The user can then browse through the entire document, reading, inserting, or removing nodes at any location. When using this approach, begin by loading XML content from a file (or some other source) into an XDocument object. The XDocument holds the entire document at once. When the XDocument class is used, the XML document is created as a series of linked .NET objects in memory.

To start building a next XML document, the user needs to create the XDocument, XElement, and XAttribute objects that constitute it. All these classes have useful constructors that allow the user to create and initialize them in one step. For example, the user can create an element and supply text content that should be placed inside using code like this:

```
XElement element = new XElement("Price", 23.99);
```

Both the XDocument and XElement class include a constructor that takes a parameter array for the last argument. This parameter array holds a list of nested nodes.

Here's an example that creates an element with three nested elements and their content:

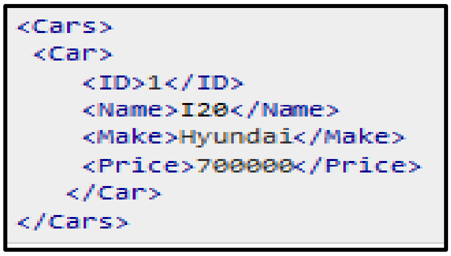
```
XElement element = new XElement("Product",
new XElement("ID", 3),
new XElement("Name", "Fresh Fruit Basket"),
new XElement("Price", 49.99) );
```

Here's the scrap of XML that this code creates:

```
<Product>
 <ID>3</ID>
 <Name>Fresh Fruit Basket</Name>
 <Price>49.99</Price>
 </Product>
```

II. SOLVED EXERCISE:

Using XmlTextWriter write the xml file in “App_Data\Cars.xml” (given in Fig 10.9).



```
<Cars>
 <Car>
    <ID>1</ID>
    <Name>I20</Name>
    <Make>Hyundai</Make>
    <Price>700000</Price>
   </Car>
</Cars>
```

Fig 10.9

```
Default.aspx.cs contents:
using System.IO;
using System.Xml;
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
       // Place the file in the App_Data subfolder of the current website.
       // The System.IO.Path class makes it easy to build the full file name.
      string file = Path.Combine(Request.PhysicalApplicationPath,
     @"App_Data\Cars.xml");
      FileStream fs = new FileStream(file, FileMode.Create);
      XmlTextWriter w = new XmlTextWriter(fs, null);
      w.WriteStartDocument();
      w.WriteStartElement("Cars");
      w.WriteComment("This file generated by the XmlTextWriter class.");
      // Write the first car.
      w.WriteStartElement("Car");
      w.WriteAttributeString("ID", "1");
      w.WriteAttributeString("Name", "I20");
      w.WriteAttributeString("Make", "Hyundai");
      w.WriteStartElement("Price");
      w.WriteString("700000");
      w.WriteEndElement();
      w.WriteEndElement();
      // Close the root element.
```

```
        w.WriteEndElement();
        w.WriteEndDocument();
        w.Close();
      }
}
```

III. LAB EXERCISES:

1) Develop a website with a GridView which displays the details of 'Staff' table. The Name and Age fields should be in a single cell under the column heading Staff details. This Staff details column should provide Name wise sorting. The rows that contain the age more than 35 should be highlighted. Add at least five rows to the Staff table which has the fields as follows:
   Name, Age, Designation, and Salary

2) Develop a web page that retrieves author details from the table 'Authors' and displays it in a GridView. The maximum records in each page can be 3. Change background color to 'green' and text color to 'red' with bold font for the header. Use data source control to fetch data. Give provisions to edit the values of FirstName and CopiesSold. The GridView should look as follows:

	Id	Name	Book Title	Genre	CopiesSold
<u>Edit</u>	1	FirstName: Shelley LastName: Pinto	Last Safari	Fiction	20

3) Create an xml file using XDocument class. The file should contain the following elements. <Cars><Detail>
   <Name>I20</Name><Make>Hyundai</Make><Price>700000</Price>
   </Detail><Detail>
   <Name>I20</Name><Make>Hyundai</Make> <Price>700000</Price>
   </Detail> </Cars>
4) Create a Website with the following features:

a. Display all the drives in a DropDownList. When the user selects the Drive its contents (files and folders) should be displayed in the ListBox.
b. When the user clicks delete button, any item selected in the ListBox should be removed.

IV. ADDITIONAL EXERCISES:

1) Develop a website to display the contents of ‘Product’ table (id, name, price, quantity) using GridView. The user should be allowed to edit the name and quantity fields only. Use appropriate validation controls to ensure that the name field is not blank and the quantity is between 1 and 20. Enable sorting of GridView based on ‘id’ and ‘name’ columns. Sample GridView is shown below:

Id	Name	Details	
1	Soap	Price: 50 Qty: 2	<u>Edit</u>

2) Create a web application as shown in figure 10.10, which accepts multiline user input and saves it to a text file under "App_Data" folder.

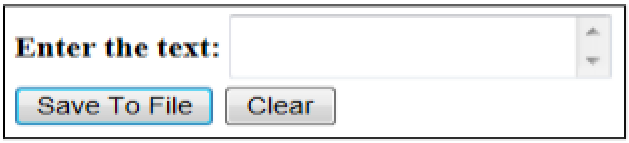


Figure 10.10

LAB NO.: 9 **Date:**

MVC & AJAX DEMO

Objectives:

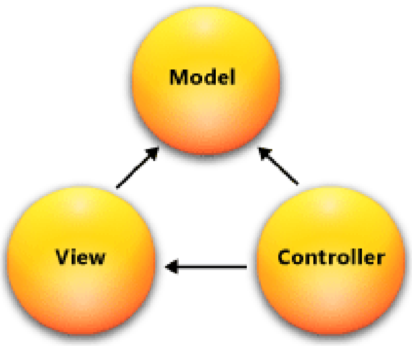
In this lab, student will be able to:

- Develop easily maintainable applications using MVC design pattern.
- Develop rich web applications that communicate with the server using asynchronous postback.

I. DESCRIPTION

The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller. The ASP.NET MVC framework provides an alternative to the ASP.NET Web Forms pattern for creating Web applications. The ASP.NET MVC framework is a lightweight, highly testable presentation framework that (as with Web Forms-based applications) is integrated with existing ASP.NET features, such as master pages and membership-based authentication. The MVC framework is defined in the **System.Web.Mvc** assembly.

MVC design pattern



The MVC framework includes the following components:

- **Models.** Model objects are the parts of the application that implement the logic for the application's data domain. Often, model objects retrieve and store model state in a database. For example, a Product object might retrieve information from a database, operate on it, and then write updated information back to a Products table in a SQL Server database.

In small applications, the model is often a conceptual separation instead of a physical one. For example, if the application only reads a dataset and sends it to the view, the application does not have a physical model layer and associated classes. In that case, the dataset takes on the role of a model object.

- **Views.** Views are the components that display the application's user interface (UI). Typically, this UI is created from the model data. An example would be an edit view of a Products table that displays text boxes, drop-down lists and checks boxes based on the current state of a Product object.
- **Controllers.** Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. For example, the controller handles query-string values and passes these values to the model, which in turn might use these values to query the database.

The MVC pattern helps the user create applications that separate the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements. The pattern specifies where each kind of logic should be located in the application. The UI logic belongs in the view. Input logic belongs in the controller. Business logic belongs in the model. This separation helps the user manage complexity when he builds an application because it enables the user to focus on one aspect of the implementation at a time. For example, one can focus on the view without depending on business logic.

The loose coupling between the three main components of an MVC application also promotes parallel development. For example, one developer can work on the view, a second developer can work on the controller logic, and a third developer can focus on the business logic in the model.

Advantages of an MVC-Based Web Application

The ASP.NET MVC framework offers the following advantages:

- It makes it easier to manage complexity by dividing an application into the model, the view, and the controller.
- It does not use view state or server-based forms. This makes the MVC framework ideal for developers who want full control over the behavior of an application.
- It uses a Front Controller pattern that processes Web application requests through a single controller. This enables the user to design an application that supports a rich routing infrastructure.

- It provides better support for test-driven development (TDD).
- It works well for Web applications that are supported by large teams of developers and for Web designers who need a high degree of control over the application behavior.

Advantages of a Web Forms-Based Web Application

The Web Forms-based framework offers the following advantages:

- It supports an event model that preserves state over HTTP, which benefits line-of-business Web application development. The Web Forms-based application provides dozens of events that are supported in hundreds of server controls.
- It uses a Page Controller pattern that adds functionality to individual pages.
- It uses view state on server-based forms, which can make managing state information easier.
- It works well for small teams of Web developers and designers who want to take advantage of a large number of components available for rapid application development.
- In general, it is less complex for application development, because the components (the **Page** class, controls, and so on) are tightly integrated and usually require less code than the MVC model.

II. SOLVED EXERCISE

The following code gives us a basic idea of working with the MVC architecture by demonstrating how to create the model, view and controller modules for the music store application by implementing the two features listed below:

- Visitors can browse Albums by Genre.
- Visitors can view a single album.

Steps to follow:

File→New→Project→VisualC# (Web) →ASP.NET Web application→ Create
Select MVC from the Templates
Right-click on the “Controllers” folder within the Solution Explorer and select “Add,” and then the “Controller…” command appears.

Select MVC 5 Controller Empty → Add → Name it as HomeController
This will create a new file, HomeController.cs, with the following code:

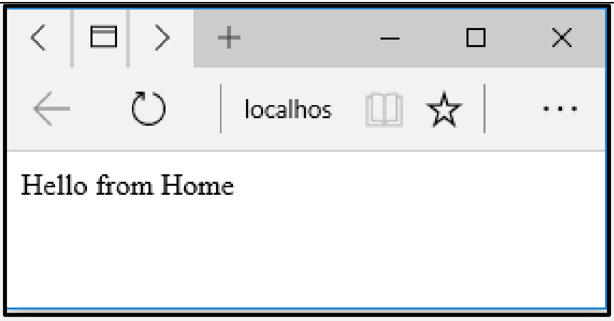
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace MvcMusicStore.Controllers
 {
    public class HomeController : Controller
   {
        //        // GET: /Home/
        public ActionResult Index()
       {
          return View();
        }
    }
  }
```

As an example the Index method can be replaced with a simple method that just returns a string. For this two changes have to be made:

- Change the method to return a string instead of an ActionResult
- Change the return statement to return “Hello from Home”

The method should now look like this:

```
public string Index( )
{
      return "Hello from Home";
 }
```



Hello from Home

Adding a StoreController:

Another controller to implement the browsing functionality of the music store has to be added. The store controller will support three scenarios:

- A listing page of the music genres in our music store

- A browse page that lists all of the music albums in a particular genre
- A details page that shows information about a specific music album

Now add a new StoreController. Just like HomeController is added, right-click on the "Controllers" folder within the Solution Explorer and choose the Add→Controller menu item

Start the StoreController implementation by changing the Index( ) method to return the string "Hello from Store.Index( )" and add similar methods for Browse( ) and Details( ) as shown below:

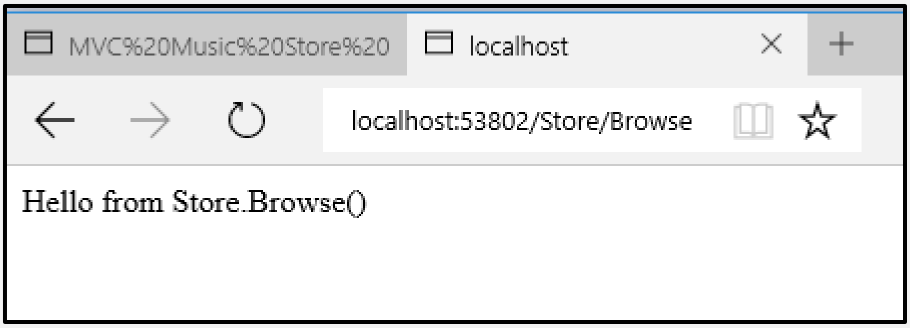
```
public class StoreController : Controller
    {
             //        // GET: /Store/
              public string Index( )
             {
                         return "Hello from Store.Index( )";
             }
              //        // GET: /Store/Browse
             public string Browse( )
             {
                         return "Hello from Store.Browse( )";
             }
             //        // GET: /Store/Details
             public string Details( )
```

```
                    {
                                return "Hello from Store.Details( )";
                    }
                }
```

Run the project again and browse the following URLs:

- /Store
- /Store/Browse
- /Store/Details

Accessing these URLs will invoke the action methods within the Controller and return string responses:

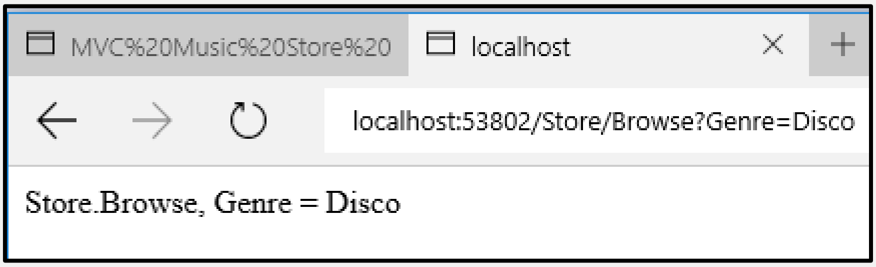


Hello from Store.Browse()

Change the Browse action method to retrieve a QueryString value from the URL. Add a “genre” parameter to the action method. ASP.NET MVC will automatically pass any QueryString or form post parameters named “genre” to the action method when it is invoked.

```
// // GET: /Store/Browse?genre=Disco
public string Browse(string genre)
{
    string message = HttpUtility.HtmlEncode("Store.Browse, Genre = " + genre);
    return message;
}
```

Now browse to /Store/Browse?Genre=Disco. The following output is obtained.



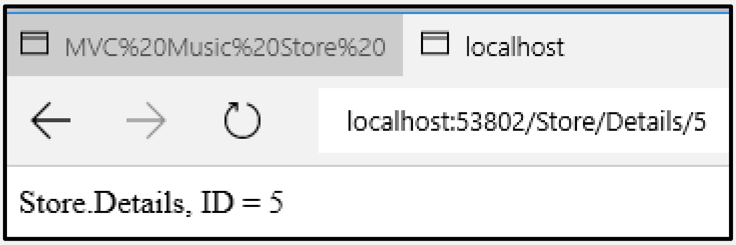
```
Store.Browse, Genre = Disco
```

Next change the Details action to read and display an input parameter named ID. Unlike previous method, the ID value is not embedded as a QueryString parameter. Instead it is embedded directly within the URL itself. For example: /Store/Details/5.

The Details( ) code must be modified as follows:

```
 // // GET: /Store/Details/5
public string Details(int id)
{
        string message = "Store.Details, ID = " + id;
        return message;
}
```

Now browse to /Store/Details/5. The following output is obtained.



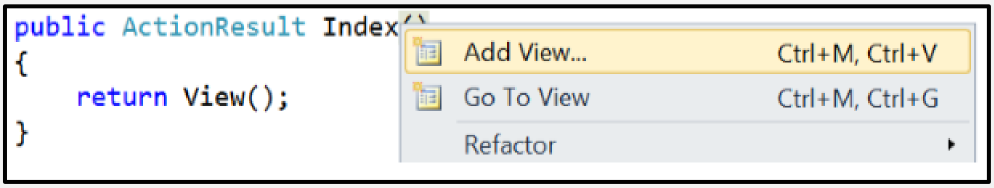
```
Store.Details, ID = 5
```

To use a view-template, change the HomeController Index method to return an ActionResult, and have it return View( ), like shown below:

```
public class HomeController : Controller
   {
```

```
                //     // GET: /Home/
               public ActionResult Index( )
               {
                         return View( );
               }
         }
```

Now add an appropriate View template to the project. To do this position the text cursor within the Index action method, then right-click and select “Add View”. This will bring up the Add View dialog:



```
public ActionResult Index()
{
    return View();
}
```

When the Add button is clicked, Visual Web Developer will create a new Index.cshtml view template in the \Views\Home directory, creating the folder if doesn’t already exist.

The name and folder location of the “Index.cshtml” file is important and follows the default ASP.NET MVC naming conventions. The directory name, \Views\Home, matches the controller - which is named HomeController. The view template name, Index, matches the controller action method which will be displaying the view.

Visual Web Developer creates and opens the “Index.cshtml” view template after the “Add” button is clicked within the “Add View” dialog. The contents of Index.cshtml are shown below.

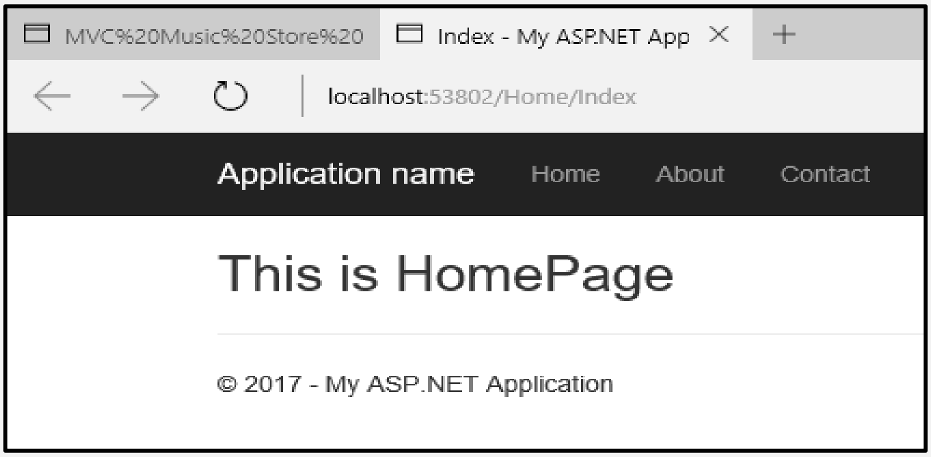
```
@{
  ViewBag.Title = "Index";
  }
 <h2>Index</h2>
```

This view is using the Razor syntax, which is more concise than the Web Forms view engine used in ASP.NET Web Forms. The first three lines set the page title using ViewBag.Title. Update the <h2> tag to say “This is the Home Page” as shown below.

```
@{
    ViewBag.Title = "Index";
```

```
        }
    <h2>This is the Home Page</h2>
```

Running the application shows that the new text is visible on the home page.

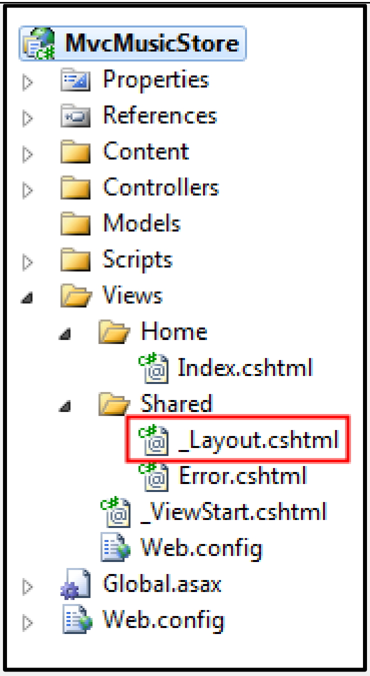


This is HomePage

© 2017 - My ASP.NET Application

Using a Layout for common site elements:

Most websites have content which is shared between many pages: navigation, footers, logo images, stylesheet references, etc. The Razor view engine makes this easy to manage using a page called _Layout.cshtml which has automatically been created inside the /Views/Shared folder.



Double-click on this file to view the contents, which are shown below.

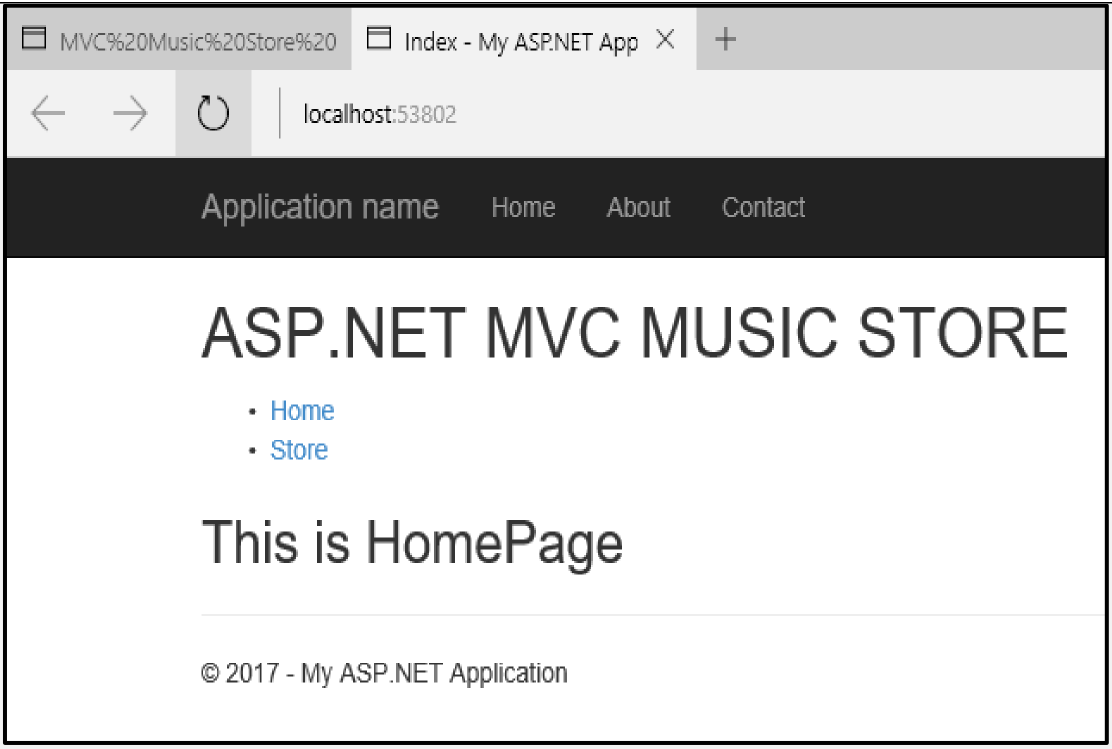
```
<!DOCTYPE html>
<html>
 <head>
<meta charset="utf-8" />
<title>@ViewBag.Title</title>
 <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css"
/>
<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")"
type="text/javascript">
</script>
 <script src="@Url.Content("~/Scripts/modernizr-1.7.min.js")"
type="text/javascript">
</script>
</head>
<body>
```

```
@RenderBody( )
</body>
</html>
```

The content from the individual views will be displayed by the @RenderBody( ) command, and any common content that needs to appear outside of that can be added to the _Layout.cshtml markup. To have a common header for the MVC Music Store with links to the Home page and Store area on all pages in the site, add that to the template directly above the @RenderBody( ) statement.

```
<body>
<div id="header">
<h1>ASP.NET MVC MUSIC STORE</h1>
<ul id="navlist">
<li class="first">
<a href="/" id="current">Home</a>
</li>
<li>
<a href="/Store/">Store </a>
</li>
</ul>
</div>
  @RenderBody( )
</body>
```

The output will be displayed as follows:



ASP.NET MVC MUSIC STORE

- Home
- Store

This is HomePage

© 2017 - My ASP.NET Application

Using a Model to pass information to the View:

A View template that just displays hardcoded HTML is not going to make a very interesting web site. To create a dynamic website, pass the information from the controller actions to the view templates. In the Model-View-Controller pattern, the term Model refers to objects which represent the data in the application. Often, model objects correspond to tables in the database.

Controller action methods which return an ActionResult can pass a model object to the view. This allows a Controller to cleanly package up all the information needed to generate a response, and then pass this information off to a View template to generate the appropriate HTML response.

Create some Model classes to represent Genres and Albums within the store. Right-click the “Models” folder within the project, choose the “Add Class” option, and name the file “Genre.cs.”

Then add a public string Name property to the class that was created:

```
public class Genre
{
      public string Name
      {
          get;
          set;
       }
 }
```

Next, follow the same steps to create an Album class (named Album.cs) that has a Title and a Genre property:

```
public class Album
   {
       public string Title
       {
           get;
           set;
        }
        public Genre Genre
        {
           get;
           set;
        }
   }
```

Change the Store Details action so that it shows the information for a single album. Add a "using" statement to the top of the StoreControllers class to include the MVC. The "usings" section of that class should now appear as below.

```
      using System;
      using System.Collections.Generic;
      using System.Linq;
      using System.Web;
      using System.Web.Mvc;
```

```
using Mvc.Models;
```

Update the Details controller action so that it returns an ActionResult rather than a string, as done in the HomeController’s Index method.

```
public ActionResult Details(int id)
```

Modify the logic to return an Album object to the view.

```
public ActionResult Details(int id)
{
        var album = new Album {  Title =  "Album "  +  id  };
        return View(album);
 }
```

Create a View template that uses our Album to generate an HTML response. Before that build the project so that the Add View dialog knows about the newly created Album class. Project can be built by selecting the Bulid →Build Mvc menu item.

Now Right-click within the Details method and select “Add View…” from the context menu. This template view will by default be generated in a \Views\Store\Index.cshtml file.

Select the Template to be “Empty” and the model class to be “Album (Mvc.Models)” in the Add View dialog. This will cause the “Add View” dialog to create a View template that expects that an Album object will be passed to it to use.

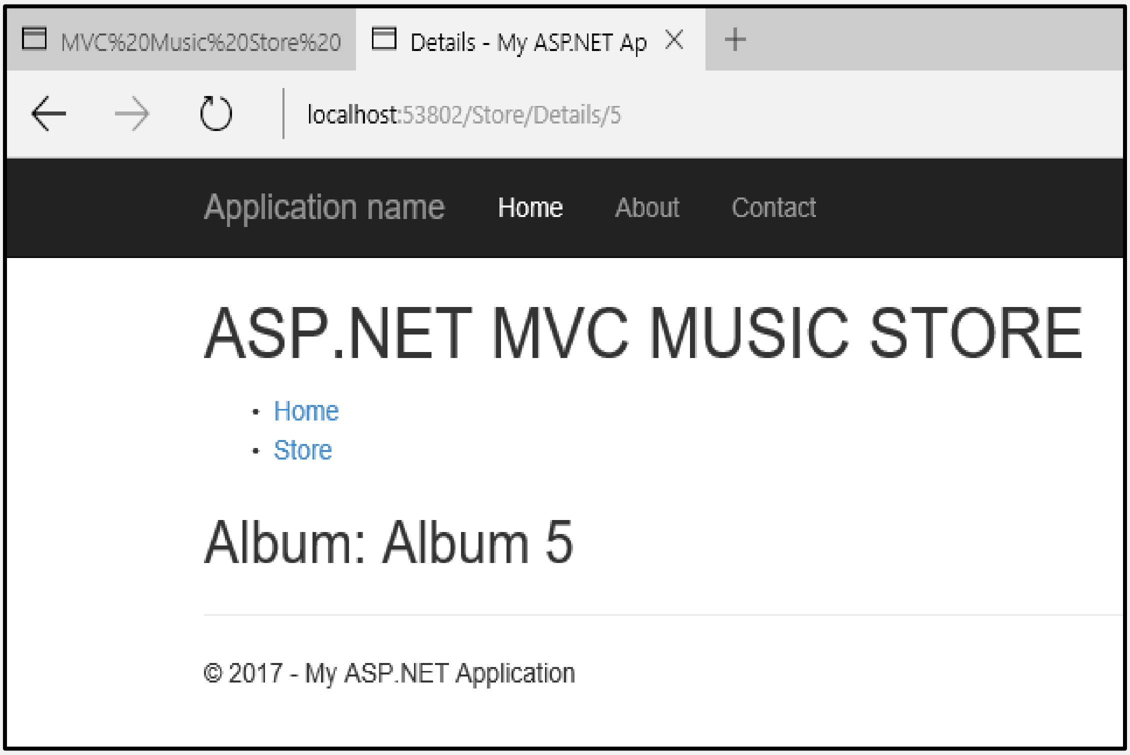
On clicking the “Add” button the \Views\Store\Details.cshtml View template will be created, containing the following code.

```
@model Mvc.Models.Album

@{
   ViewBag.Title = "Details";
   }

<h2>Details</h2>
```

Update the <h2> tag so it displays the Album's Title property by modifying that line to appear as follows.

```
<h2>Album: @Model.Title</h2>
```

Now re-run the project and visit the /Store/Details/5 URL. The following output is obtained.



ASP.NET MVC MUSIC STORE

- Home
- Store

Album: Album 5

© 2017 - My ASP.NET Application

Make a similar update for the Store Browse action method. Update the method so that it returns an ActionResult, and modify the method logic so that it creates a new Genre object and returns it to the View.

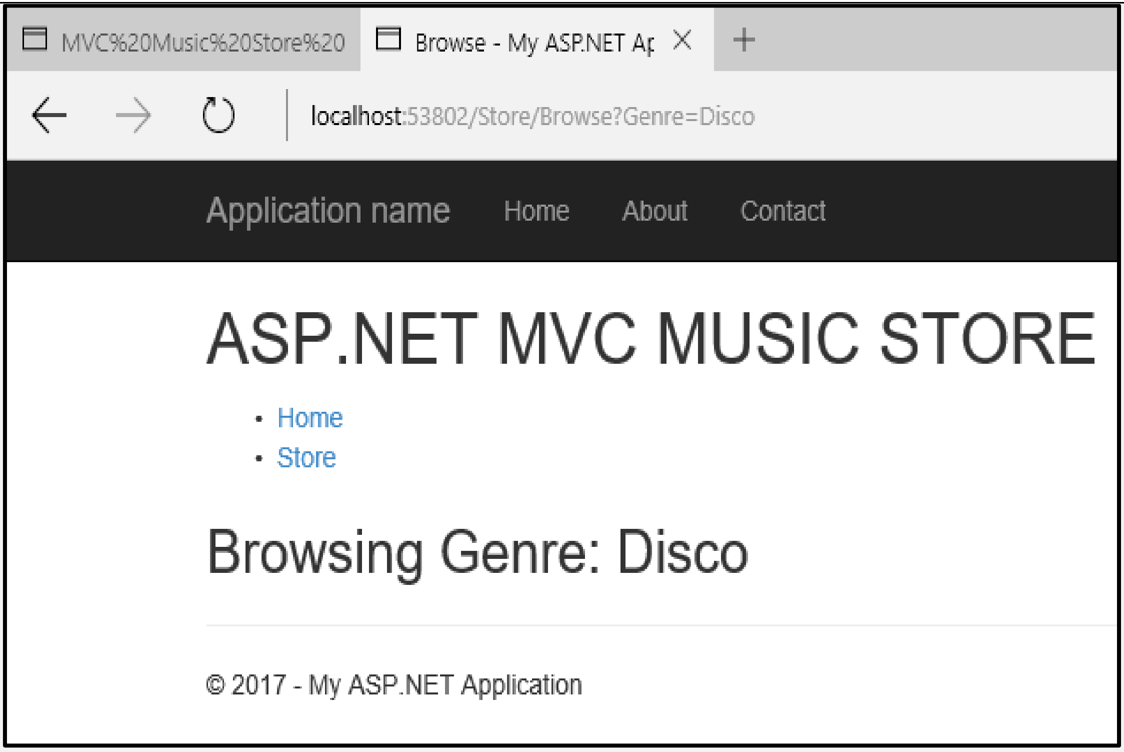
```
public ActionResult Browse(string genre)
{
        var genreModel = new Genre { Name = genre };
        return View(genreModel);
}
```

Right-click in the Browse method and select “Add View…” from the context menu. Select the Template to be “Empty” and the model class to be “Genre (Mvc.Models)” in the Add View dialog.

Update the <h2> element in the view code (in /Views/Store/Browse.cshtml) to display the Genre information.

```
@model Mvc.Models.Genre

@{

    ViewBag.Title = "Browse";

   }

<h2>Browsing Genre: @Model.Name</h2>
```

Re-run the project and browse to the /Store/Browse?Genre=Disco URL. The output is as follows:


MVC%20Music%20Store%20

Browse - My ASP.NET Ap

localhost:53802/Store/Browse?Genre=Disco

Application name Home About Contact

ASP.NET MVC MUSIC STORE

- Home
- Store

Browsing Genre: Disco

© 2017 - My ASP.NET Application

Update the Store Index action method and view to display a list of all the Genres in the store. Use a List of Genres as the model object, rather than just a single Genre. (In StoreController class)

```
public ActionResult Index( )
{
    var genres = new List<Genre> { new Genre { Name = "Disco" },
    new Genre { Name = "Jazz" }, new Genre { Name = "Rock" } };
    return View(genres);
}
```

Right-click in the Store Index action method and select Add View as before, select Genre as the Model class, and press the Add button.

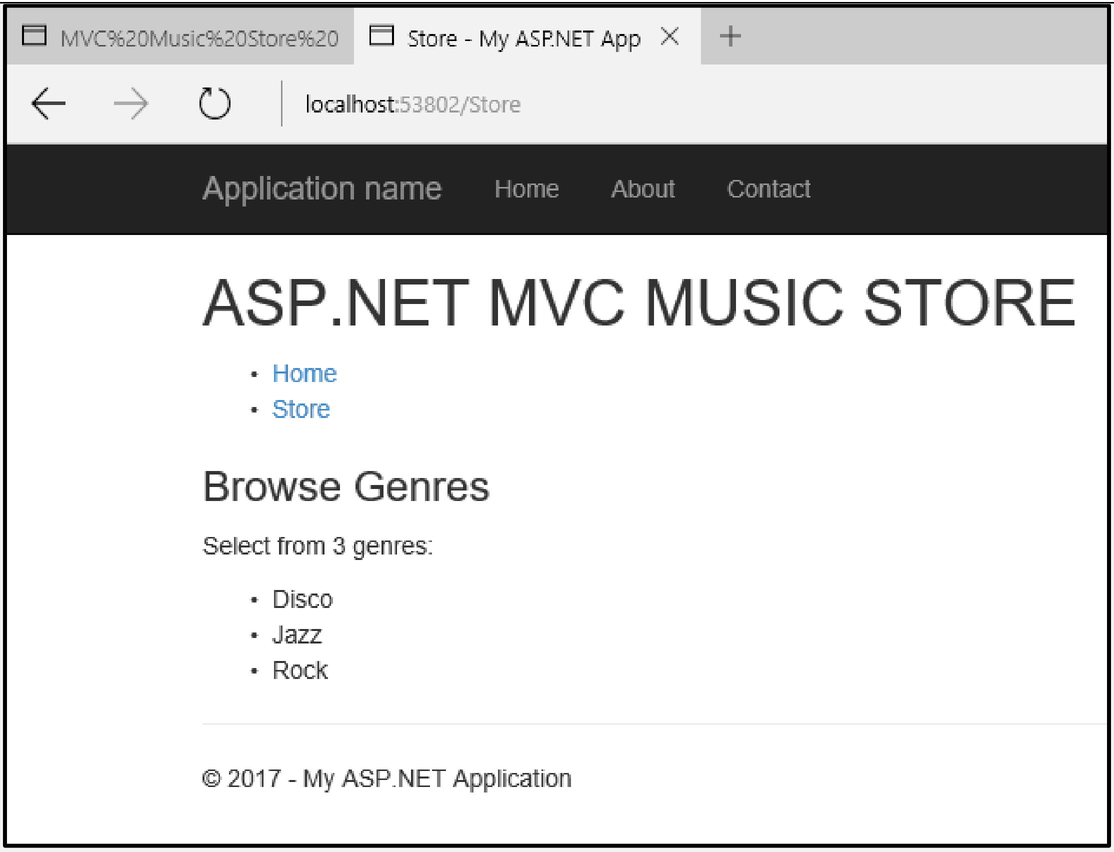
Change the @model declaration to indicate that the view will be expecting several Genre objects rather than just one. Change the first line of /Store/Index.cshtml to read as follows:

```
@model IEnumerable<MvcMusicStore.Models.Genre>
```

Next, loop through the Genre objects in the model as shown in the completed view code below.

```
@model IEnumerable<Mvc.Models.Genre>
   @{
      ViewBag.Title = "Store";
      }
   <h3> Browse Genres </h3>
   <p>Select from @Model.Count( ) genres:</p>
   <ul>
      @foreach (var genre in Model)
      {
      <li>@genre.Name</li>
      }
   </ul>
```

Run the application and browse to /Store, both the count and list of Genres is displayed as follows:



ASP.NET MVC MUSIC STORE

- Home
- Store

Browse Genres

Select from 3 genres:

- Disco
- Jazz
- Rock

© 2017 - My ASP.NET Application

Adding Links between pages:

ASP.NET MVC includes HTML Helper methods which are available from the View template code to perform a variety of common tasks. The Html.ActionLink( ) helper method is a particularly useful one, and makes it easy to build HTML <a> links and takes care of annoying details like making sure URL paths are properly URL encoded.

The link text and the Action method to go to, when the hyperlink is clicked on the client has to be supplied. For example, it is possible to link to "/Store/" Index( ) method on the Store Details page with the link text "Go to the Store Index" using the following call:

@Html.ActionLink("Go to the Store Index", "Index")

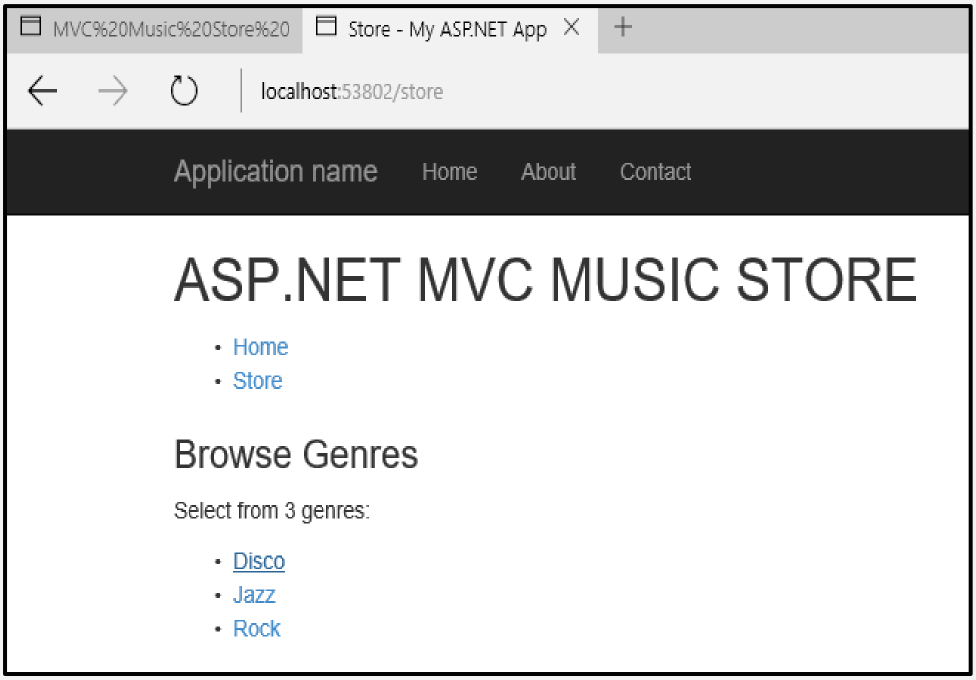
The links to the Browse page will require to pass a parameter, though, so use another overload of the Html.The actionlink method that takes three parameters:

- Link text, which will display the Genre name
- Controller action name (Browse)
- Route parameter values, specifying both the name (Genre) and the value (Genre name)

Putting that all together, the links to the Store Index view is written as follows:

```
<ul>
    @foreach (var genre in Model)
    {
        <li>@Html.ActionLink(genre.Name, "Browse",
        new { genre = genre.Name })</li>
    }
 </ul>
```

Run the project again and access the /Store/ URL to see a list of genres. Each genre is a hyperlink – when clicked it will navigate the user to the /Store/Browse?genre=[genre] URL.



ASP.NET MVC MUSIC STORE

- Home
- Store

Browse Genres

Select from 3 genres:

- Disco
- Jazz
- Rock

III. LAB EXERCISES:

1) Create a simple question-asking site like stack overflow, Quora, etc using ASP.NET MVC framework that allows logged users to ask a new question and answer existing questions. The logged users should also be allowed to upvote or downvote a question. Upvoting and down voting should happen asynchronously without reloading the entire web page.

2) Create an ASP.NET MVC application for “Manipal Travels”. The application can be accessed by Administrator and Anonymous user.
    - The administrator should be able to add new place and buses between two places.
    - Anonymous users should be able to search for available seats between two places asynchronously without reloading the entire web page.
    - Anonymous users should be allowed to book available seats by providing the name and phone number.

3) Create an e-auction site using ASP.NET MVC framework. The site should allow users to view different items available for bidding along with its due date. Only logged users can bid for an item. On the due date, a user with the highest bid will get the item. The administrator should be able to create additional items and view/update all existing items.

4) Create an ASP.NET MVC application for MIT. It should include functionality such as student admission, course creation, and instructor assignments. Users can view and update student, course, and instructor information.

IV. ADDITIONAL EXERCISES:

1) Create a simple movie listing application using ASP.NET MVC framework that supports creating, editing, searching and listing movies from a database. All data-entry scenarios should include validation to ensure that the data stored in the database is correct.

2) Create an event management MVC application that allows users to create/edit/delete and list events.

LAB NO.: 10 **Date:**

HTML5, CSS AND JAVASCRIPT DEMO

Objectives:

In this lab, student will be able to:

- Develop HTML5 web pages
- Familiarize with Cascading Style Sheets
- Embed the JavaScript code in HTML5 pages

I. DESCRIPTION

HTML5 – Hyper Text Markup Language Version 5

HTML5 is the 5th and newest version of HTML standard, providing new features like rich media support, interactive web applications etc.

The most interesting HTML5 elements are:

1. Semantic elements like <header>, <footer>, <article> and <section>
2. Attributes of form elements like number, date, time, calendar and range.
3. Graphic elements like <svg> and <canvas>
4. Multimedia elements like <audio> and <video>

There are several Application Programming Interfaces too in HTML5 like HTML Geolocation, HTML Drag and Drop, HTML Web Workers etc.

Several elements of HTML4 have been removed in HTML5 like <big>, <center>, <font>, <frame>, <frameset>, <strike> etc.

To indicate that your HTML content uses HTML5, simply add <!DOCTYPE html> on top of the html code.

Procedure to create and HTML document:

In notepad type the necessary code & save with the file name mentioned with .html or .htm extension.

Example:

```
<html>
<head>
<title> My First Page </title>
</head>
<body>
<h1> Hello </h1>
<h2> Welcome to Internet Technologies Lab </h2>
</body>
</html>
```

HTML5 Elements

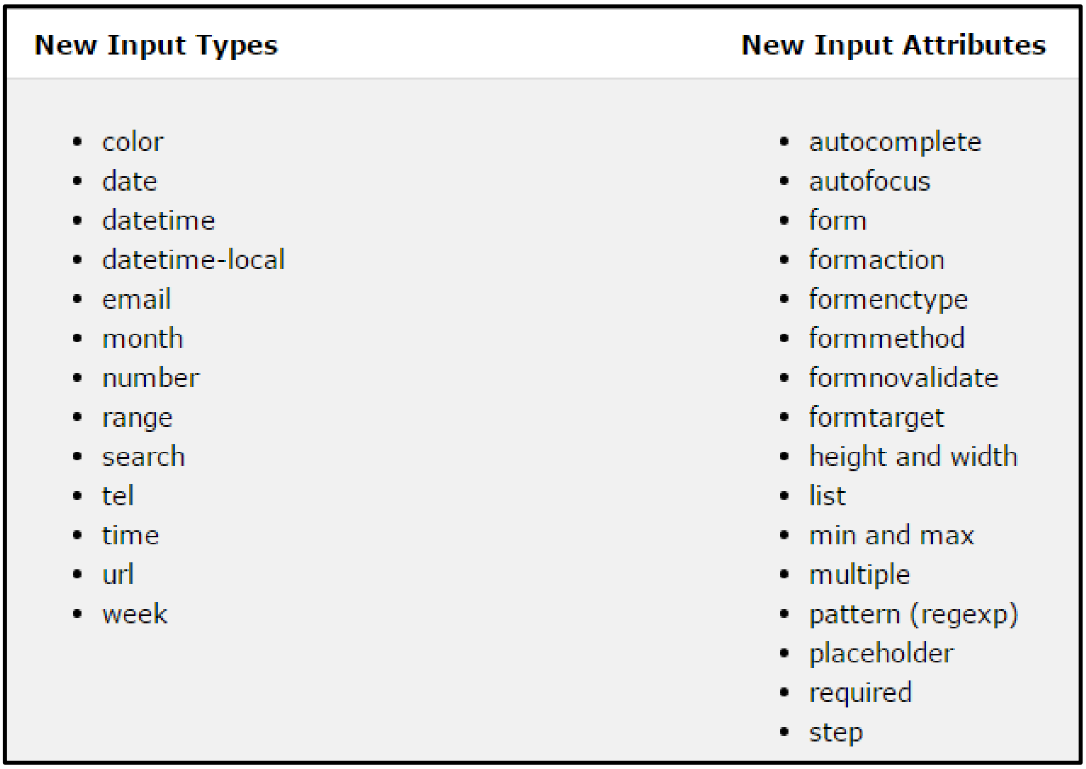
HTML5 offers new elements for better document structure. The below given table gives a brief description on few HTML5 elements.

TAG	**DESCRIPTION**
<article>	Defines an article in a document
<dialog>	Defines a dialog box or window
<header>	Defines a header for a document or a section
<footer>	Defines a footer for a document or a section
<nav>	Defines navigation links
<time>	Defines a date/time
<output>	Defines the result of a calculation
<canvas>	Draw graphics, on the fly, via scripting(JavaScript)
<audio>	Defines sound content

<source>	Defines multiple media resources for media elements
<video>	Defines video or movie

Figure 1.1 Few HTML5 elements

HTML5 Input types and Attributes



New Input Types

- color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

New Input Attributes

- autocomplete
- autofocus
- form
- formaction
- formenctype
- formmethod
- formnovalidate
- formtarget
- height and width
- list
- min and max
- multiple
- pattern (regexp)
- placeholder
- required
- step

Figure 1.2 HTML5 input types and attributes

The above figure lists the new input types and attributes of HTML5.

HTML5 Events

On visiting a website the user perform actions like clicking on links or image, hover over things etc. These are considered to be examples for Events.

Event handlers are developed to handle these events and this can be done using a scripting language like JavaScript, VBScript etc wherein event handlers are specified as a value of event tag attribute.

The following attributes (very few) can be used to trigger any **javascript** or **vbscript** code given as value, when there is any event occurs for any HTM5 element.

Attribute	**Value**	**Description**
offline	script	Triggers when the document goes offline
onchange	script	Triggers when an element changes
onclick	script	Triggers on a mouse click
oncontextmenu	script	Triggers when a context menu is triggered
ondrag	script	Triggers when an element is dragged
onerror	script	Triggers when an error occur
onfocus	script	Triggers when the window gets focus
onformchange	script	Triggers when a form changes
onload	script	Triggers when the document loads
onmousedown	script	Triggers when a mouse button is pressed
onpause	script	Triggers when a media data is paused
onselect	script	Triggers when an element is selected
onsubmit	script	Triggers when a form is submitted

Figure 1.3 HTML5 event attributes

HTML5 Canvas

The HTML <canvas> element is used to draw graphics, on the fly, via JavaScript. The <canvas> element is only a container for graphics. The user must use JavaScript to actually draw the graphics. Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content. The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Note: Always specify an Id attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas. To add a border, use the style attribute.

Example: To draw a circle

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
</script>
 </body>
</html>
```

Output

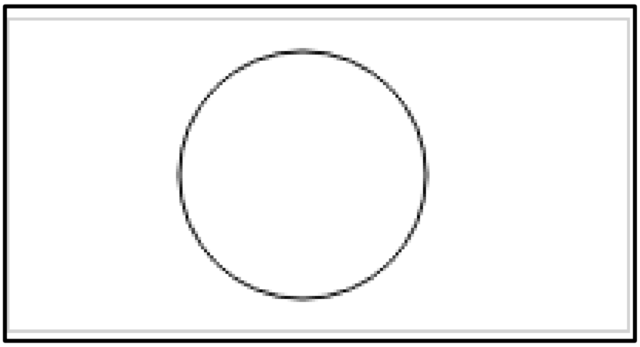


Figure 1.4

HTML5 Web Forms

The HTML <form> element defines a form that is used to collect user input. Form elements consist of various input elements like text field, check boxes, radio buttons, submit buttons etc.

<form>

</form>

Input Element

It is the most important form element and can be displayed in several ways, depending on the type attribute.

<input type = “text”>	Defines a one line text input field
<input type = “radio”>	Defines a radio button
<input type = “submit”>	Defines a submit button

Action attribute

The action attribute defines the action to be performed when the form is submitted. Usually the form data is sent to a web page on the server when the user clicks on the submit button.

<form action = “/action_page.aspx”>

// action_page.aspx contains a server side script that handles the form data.

If the action attribute is omitted, the action is set to the current page.

Method attribute

The method attribute specifies the HTTP method (GET or POST) to be used when submitting the form data.

<form action = “/action_page.aspx” method=”get”>

The default method when submitting form data is GET. When GET is used the submitted form data will be visible in the page address field. Therefore it must not be used when sending sensitive information.

Use POST method if the form data contains sensitive or personal information. It does not display the submitted form data in the page address field. It has no size limitations and can be used to send large amounts of data.

CSS – Cascading Style Sheet

CSS is a stylesheet language used for describing the presentation of a document written in a markup language ie it describes the style of a web document including the layout, design and display variations for various displays. CSS can be applied to a web document in 3 ways.

1) Inline style – Right next to the text it decorates, by using style attribute.
   <h1 style = “color : blue ;”> Hello </h1>

2) Internal style – At the top of the web page document, using <style> element in <head>
   ```
   <head>

   <style>

   h1  { color : blue ;}

   </style> </head>
   ```

3) External style – in a separate file
   ```
   <head>

   <link rel=”stylesheet” href = “style.css”>

   </head>
   ```

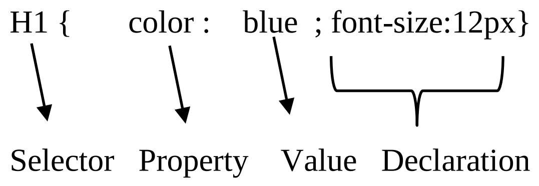
   style.css

   ```
   h1  { color : blue ;}
   ```

The style definitions are usually saved in an external stylesheet since changing one single file can help in redesigning the entire web document with new look and feel.

CSS syntax

A CSS rule set consists of a selector and a declaration block. The selector points to the HTML element to be styled. The declaration block contains one or more declarations separated by semicolons.



CSS Selectors are used to “find” or select HTML elements based on their element name, id, class, attribute etc. The element selector selects the elements based on the element name. The id selector uses the id attribute of an HTML element to select a specific element. The id of an element should be unique within a page. To select an element with a specific id, write a # character followed by the id of the element.

```
#para1{

 text-align: center;

color:red; }
```

The class selector selects the elements with a specific class attribute. To select elements with a specific class, write a period (.) character, followed by the name of the class.

```
.center {

text-align: center;

color:red; }
```

<u>JavaScript</u>

JavaScript is a light weight and interpreted programming language. It is a scripting language that is commonly used for the client side web development. It makes the HTML pages more dynamic and interactive.

It can be used to put dynamic text in to HTML page (Eg: document.write("<h1>" + name + "</h1>"); // write variable text in to HTML page), react to events, validate data, create cookies etc.

<u>Syntax</u>

It can be implemented using JavaScript statements that are placed within the <script>….</script> HTML tags in a web page. The <script> tags, containing the JavaScript code can be placed anywhere within the web page, but normally recommended to place within the <head> tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as script. The script tag have mainly two attributes language and type, specifying the scripting language used.

To select an HTML element, JavaScript very often use the document.getElementById(id) method. The word document.write is a standard JavaScript command for writing output to a page.

```
<script language=”javascript” type=”text/javascript”>

document.getElementById("demo").innerHTML = "Hello JavaScript!";

</script>
```

II. SOLVED EXERCISE:

Develop an HTML5 program to validate the credentials with appropriate internal styling with the help of CSS and JavaScript.

Program:

<u>Login.html</u>

```
<!DOCTYPE html>
<html>
<head>
<title>  Login page  </title>
<style>
body {
   background-color: lightblue;
}

h1 {
   color: white;
   text-align: center;
}

p {
   font-family: verdana;
   font-size: 20px;
}
</style>
<script language="javascript">
function check(form)        /*function to check userid & password*/
{
  /*the following code checkes whether the entered userid and password are
matching*/
 if(form.userid.value == "myuserid" && form.pswrd.value == "mypswrd")
  {
   window.open("https://www.google.com", "_blank");
/*opens the target page while Id & password matches*/
  }
 else
 {
  alert("Error Password or Username")    /*displays error message*/
  }
}
</script>
```

```
</head>
<body>
<h1> Validate Credentials </h1>
<form name="login">
Username<input type="text" name="userid"/>
Password<input type="password" name="pswrd"/>
<input type="button" onclick="check(this.form)" value="Login"/>
<input type="reset" value="Cancel"/>
</form>
</body>
</html>
```

Output

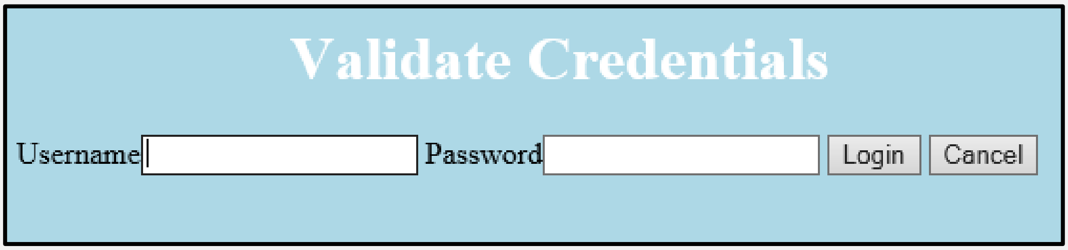


Figure 1.5

III. LAB EXERCISE:

1) Create an HMTL5 document to get an HTML5 element’s position on the web page with the help of CSS and JavaScript function.
2) Write a JavaScript program to Wish a user at different hours of a day. Use appropriate dialog boxes for wishing the user. Display the dynamic clock on the web page. Make use of CSS and HTML5 elements for creative and attractive designs.
3) Create an HTML5 document that displays a bouncing ball. Use HTML5 elements, CSS and JavaScript functions.
4) Create a HTML5 form describing your department with all the possible attributes: <INPUT> with all TYPEs, <SELECT>, <TEXTAREA>, <IMAGE>etc.

ADDITIONAL EXERCISES:

1) Develop a color-picker using HTML5 elements, CSS and JavaScript functions.

Create an animation of rain using HTML5 canvas element. Apply appropriate usage of CSS and JavaScript function to develop the animation

REFERENCES

1. Achyut Godbole, Atul Kahate, "*Web Technologies*", McGraw Hill 3rd edition, 2013.
2. Matthew MacDonald, "*Beginning ASP.NET 4.5 in C#",* Apress, 2012.
3. Jason N. Gaylord, Christian Wenz, Pranav Rastogi, Todd Miranda, Scott Hanselman, *"Professional ASP.NET 4.5 in C# and VB*" , Wrox, 2013.
4. Stephen Walther, Nate Scott Dudek, "*ASP.NET 4.5 Unleashed*", Pearson Education Inc., 2013.
5. Adam Freeman, Matthew MacDonald, Mario Szpuszta, "*Pro ASP.NET 4.5 in C#",* Apress, 2013.
6. Elliotte Rusty Harold, W. Scott Means, "*XML In a nutshell*", O'Reilly 3rd edition, 2005.
7. https://www.w3schools.com/html/html5_intro.asp
8. https://www.w3schools.com/css/
9. https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx
10. https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx
11. https://www.tutorialspoint.com/javascript/