Sub:	Programming Using C#							
Date:	22/07/23	Duration:	90 min’s	Max Marks:	50	Sem:	IV	

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

PART I

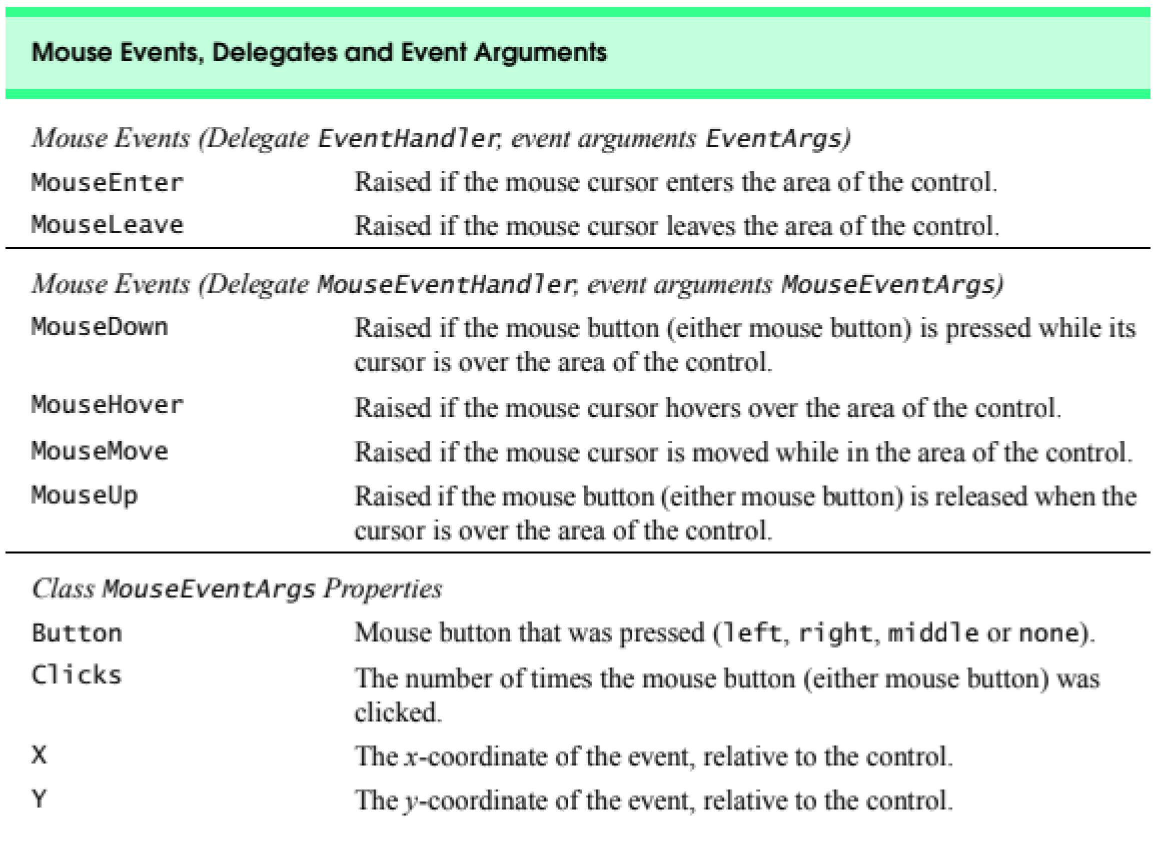
1 **Write a program in C# to change fonts using checkboxes**

Ans :

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp5 {

public partial class Form1 : Form {

	public Form1()
	{
		InitializeComponent();
	}

	private void Form1_Load(object sender, EventArgs e)
	{
		// Creating and setting the properties of label
		Label l = new Label();
		l.Text = "Select language:";
		l.AutoSize = true;
		l.Location = new Point(233, 111);
		l.Font = new Font("Bradley Hand ITC", 12);
		// Adding label to form
		this.Controls.Add(l);

		// Creating and setting the properties of CheckBox
		CheckBox Mycheckbox = new CheckBox();
		Mycheckbox.Height = 50;
		Mycheckbox.Width = 100;
		Mycheckbox.Location = new Point(229, 136);
		Mycheckbox.Text = "C#";
		Mycheckbox.BackColor = Color.LightPink;
		Mycheckbox.ForeColor = Color.DarkGreen;
		Mycheckbox.Name = "First_Check_Box";
		Mycheckbox.Font = new Font("Bradley Hand ITC", 12);

		// Adding checkbox to form
		this.Controls.Add(Mycheckbox);

		// Creating and setting the properties of CheckBox
		CheckBox Mycheckbox1 = new CheckBox();
		Mycheckbox1.Height = 50;
		Mycheckbox1.Width = 100;
		Mycheckbox1.Location = new Point(250, 198);
		Mycheckbox1.Text = "Ruby";
		Mycheckbox1.BackColor = Color.LightGreen;
		Mycheckbox1.ForeColor = Color.DeepPink;
		Mycheckbox1.Name = "Second_Check_Box";
		Mycheckbox1.Font = new Font("Bradley Hand ITC", 12);

		// Adding checkbox to form
```

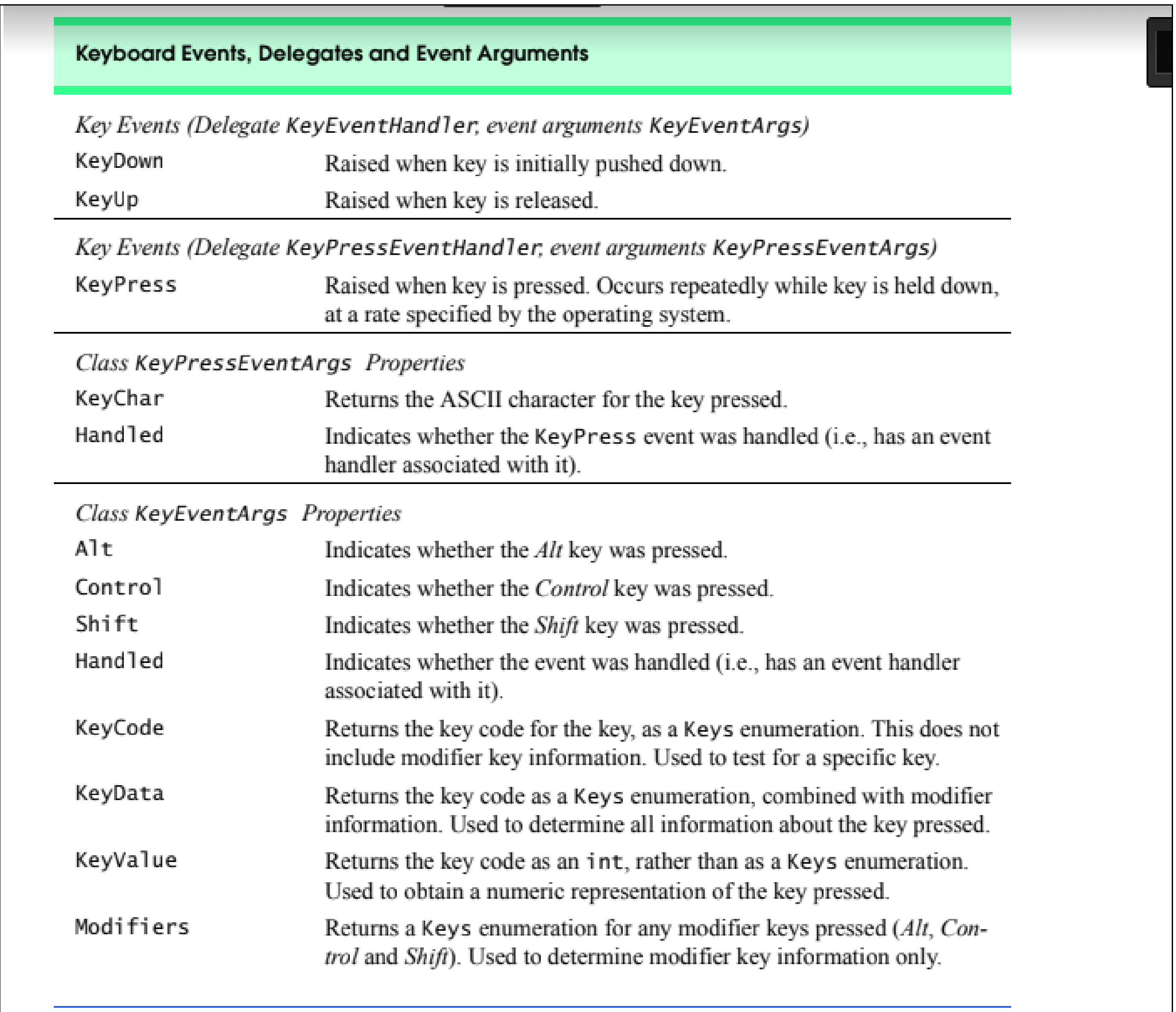
```
                    this.Controls.Add(Mycheckbox1);
          }
}
}
```

2 **Explain Keyboard and Mouse event handling.**



Mouse Events, Delegates and Event Arguments	
Mouse Events (Delegate `EventHandler`, event arguments `EventArgs`)	
`MouseEnter`	Raised if the mouse cursor enters the area of the control.
`MouseLeave`	Raised if the mouse cursor leaves the area of the control.
Mouse Events (Delegate `MouseEventHandler`, event arguments `MouseEventArgs`)	
`MouseDown`	Raised if the mouse button (either mouse button) is pressed while its cursor is over the area of the control.
`MouseHover`	Raised if the mouse cursor hovers over the area of the control.
`MouseMove`	Raised if the mouse cursor is moved while in the area of the control.
`MouseUp`	Raised if the mouse button (either mouse button) is released when the cursor is over the area of the control.
Class `MouseEventArgs` Properties	
`Button`	Mouse button that was pressed (`left`, `right`, `middle` or `none`).
`Clicks`	The number of times the mouse button (either mouse button) was clicked.
`X`	The *x*-coordinate of the event, relative to the control.
`Y`	The *y*-coordinate of the event, relative to the control.

Fig. 12.35 Mouse events, delegates and event arguments.



Keyboard Events, Delegates and Event Arguments

Key Events (Delegate `KeyEventHandler`, event arguments `KeyEventArgs`)	
`KeyDown`	Raised when key is initially pushed down.
`KeyUp`	Raised when key is released.
Key Events (Delegate `KeyPressEventHandler`, event arguments `KeyPressEventArgs`)	
`KeyPress`	Raised when key is pressed. Occurs repeatedly while key is held down, at a rate specified by the operating system.
Class `KeyPressEventArgs` Properties	
`KeyChar`	Returns the ASCII character for the key pressed.
`Handled`	Indicates whether the `KeyPress` event was handled (i.e., has an event handler associated with it).
Class `KeyEventArgs` Properties	
`Alt`	Indicates whether the *Alt* key was pressed.
`Control`	Indicates whether the *Control* key was pressed.
`Shift`	Indicates whether the *Shift* key was pressed.
`Handled`	Indicates whether the event was handled (i.e., has an event handler associated with it).
`KeyCode`	Returns the key code for the key, as a `Keys` enumeration. This does not include modifier key information. Used to test for a specific key.
`KeyData`	Returns the key code as a `Keys` enumeration, combined with modifier information. Used to determine all information about the key pressed.
`KeyValue`	Returns the key code as an `int`, rather than as a `Keys` enumeration. Used to obtain a numeric representation of the key pressed.
`Modifiers`	Returns a `Keys` enumeration for any modifier keys pressed (*Alt*, *Control* and *Shift*). Used to determine modifier key information only.

Fig. 12.38 Keyboard events, delegates and event arguments.

PART II

3 **Write a C# program to display hidden text in a password box**

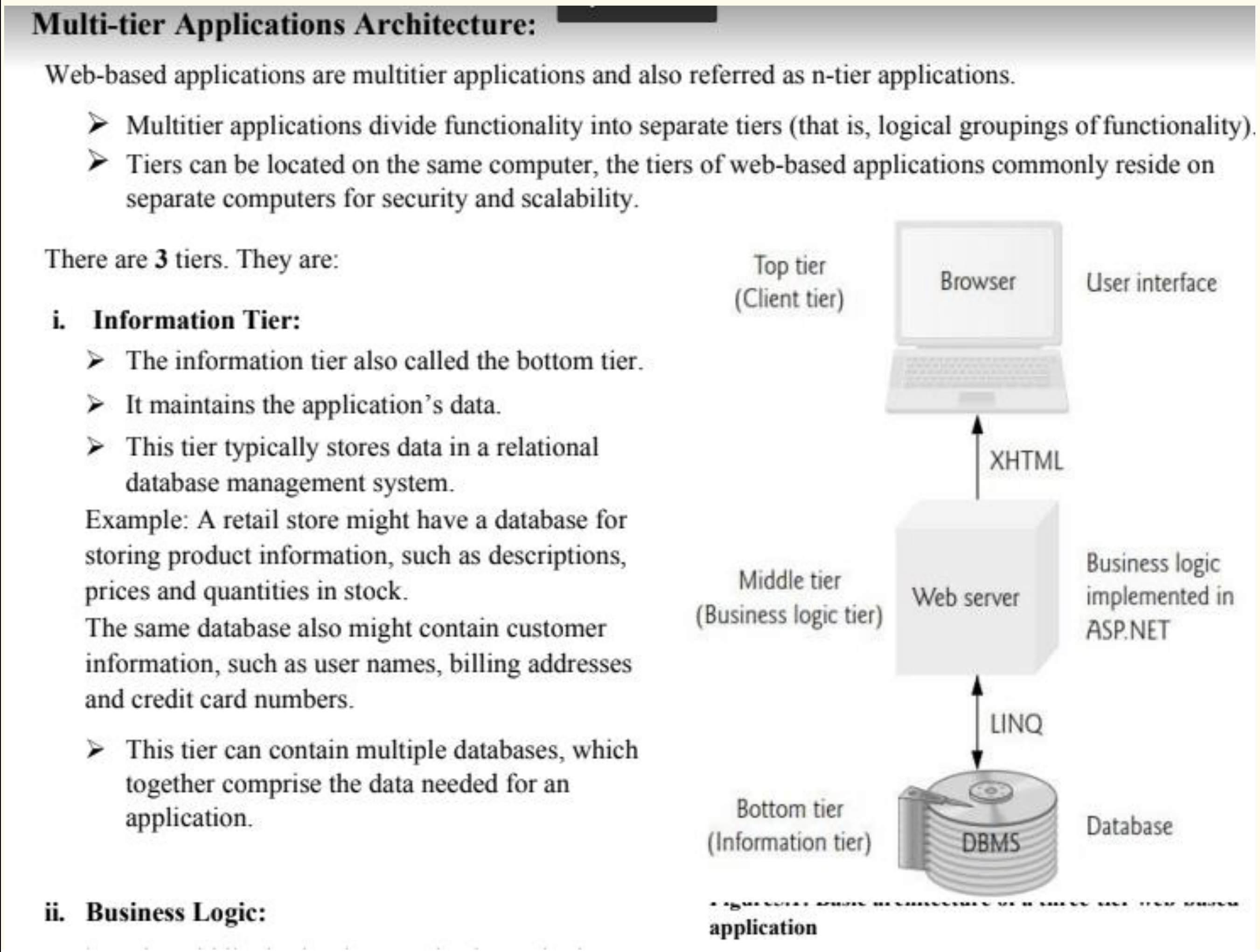
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Csharp_And_MySQL
{
   public partial class Show_Hide_Password : Form
   {
        public Show_Hide_Password()
        {
            InitializeComponent();
        }

        private void checkBox_Show_Hide_CheckedChanged(object sender, EventArgs e)
        {
            if(checkBox_Show_Hide.Checked)
            {
                textBox_Password.UseSystemPasswordChar = true;
            }
            else
            {
                textBox_Password.UseSystemPasswordChar = false;
            }
        }
    }
}
```

4 **Write a C# program to add, removes and clear ListBox items**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Csharp_And_MySQL
{
    public partial class Csharp_ListBox : Form
    {
        public Csharp_ListBox()
        {
            InitializeComponent();
        }

        private void BTN_ADD_Click(object sender, EventArgs e)
        {
            // add item to listbox from textbox
            listBox1.Items.Add(textBox1.Text);

           // add item to listbox from combobox
            listBox1.Items.Add(comboBox1.SelectedItem.ToString());
        }

        // button move selected listbox item up
        private void BTN_UP_Click(object sender, EventArgs e)
        {

            int i = listBox1.SelectedIndex;

            string item = listBox1.SelectedItem.ToString();
            if(i > 0)
            {
                listBox1.Items.RemoveAt(i);
                listBox1.Items.Insert(i - 1, item);
                listBox1.SetSelected(i - 1, true);
            }

        }

        // button move selected listbox item down
        private void BTN_DOWN_Click(object sender, EventArgs e)
        {
            int i = listBox1.SelectedIndex;

            string item = listBox1.SelectedItem.ToString();
            if (i < listBox1.Items.Count -1 )
            {
                listBox1.Items.RemoveAt(i);
                listBox1.Items.Insert(i + 1, item);
                listBox1.SetSelected(i + 1, true);
            }
        }

    }
}
```

PART III

5 **Discuss about multitier application architecture**



Multi-tier Applications Architecture:

Web-based applications are multitier applications and also referred as n-tier applications.

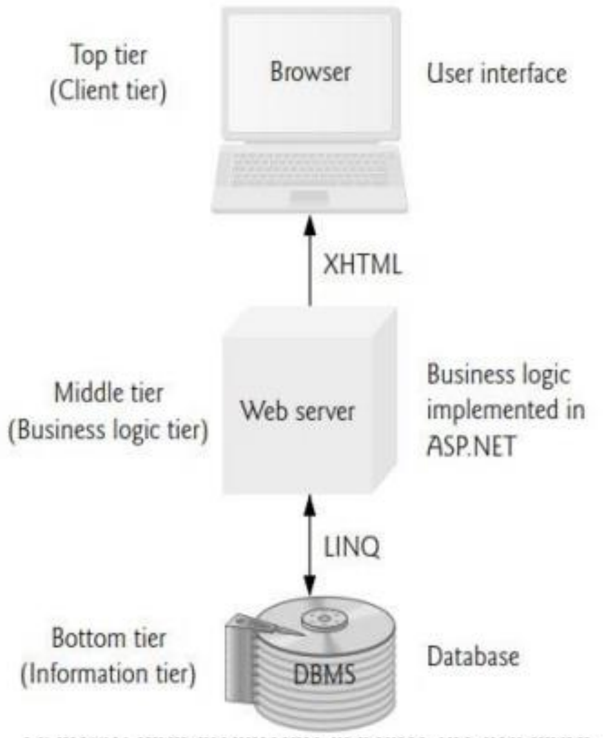
- Multitier applications divide functionality into separate tiers (that is, logical groupings of functionality).
- Tiers can be located on the same computer, the tiers of web-based applications commonly reside on separate computers for security and scalability.

There are **3** tiers. They are:

i. **Information Tier:**
- The information tier also called the bottom tier.
- It maintains the application's data.
- This tier typically stores data in a relational database management system.

Example: A retail store might have a database for storing product information, such as descriptions, prices and quantities in stock.
The same database also might contain customer information, such as user names, billing addresses and credit card numbers.

- This tier can contain multiple databases, which together comprise the data needed for an application.



application

ii. **Business Logic:**

- The middle tier implements business logic,
  controller logic and *presentation logic* to control interactions between the application's clients and its data.
- The middle tier acts as an intermediary between data in the information tier and the application's clients.
- The middle-tier **controller logic** processes client requests (such as requests to view a product catalog) and retrieves data from the database.
- The middle-tier **presentation logic** then processes data from the information tier and presents the content to the client.
- Web applications typically present data to clients as web pages.
- Business logic in the middle tier enforces business rules and ensures that data is reliable before the server application updates the database or presents the data to users.
- Business rules dictate how clients can and cannot access application data, and how applications process data.

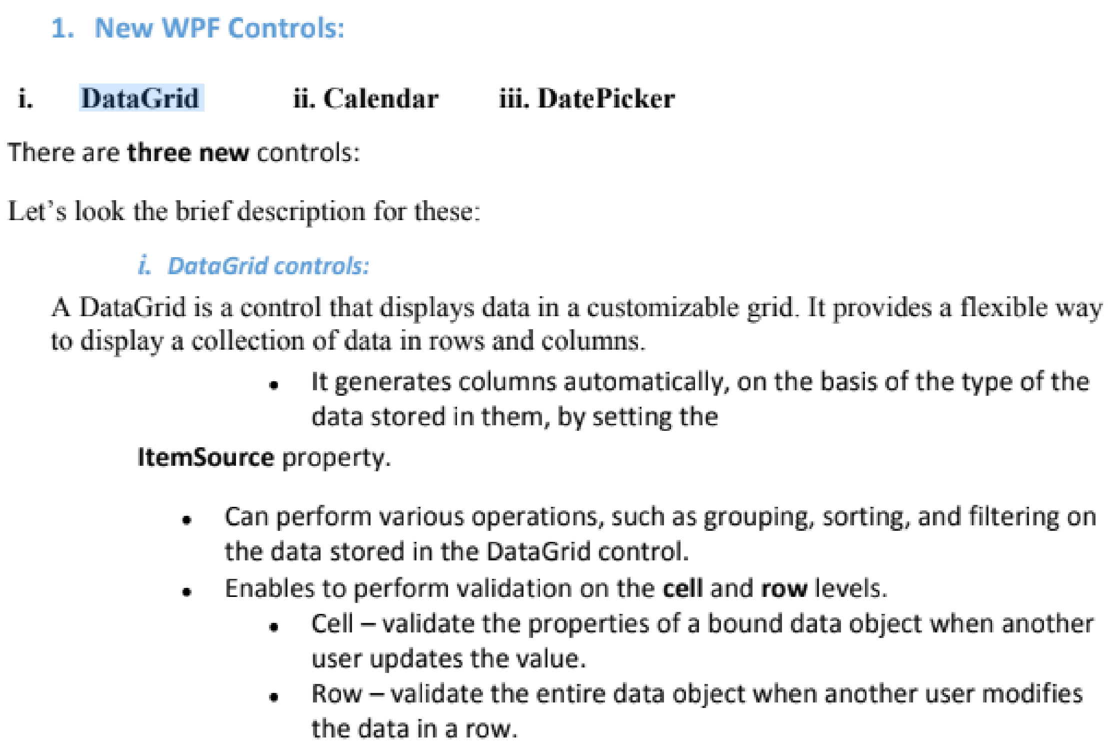
Example: A business rule in the middle tier of a retail store's web-based application might ensure that all product quantities remain positive.

A client request to set a negative quantity in the bottom tier's product information database would be rejected by the middle tier's business logic.

iii. Client Tier:

- The client tier, or top tier, is the application's user interface, which gathers input and displays output.
- Users interact directly with the application through the user interface (typically viewed in a web browser), keyboard and mouse.
- In response to user actions (Example: clicking a hyperlink), the client tier interacts with the middle tier to make requests and to retrieve data from the information tier.
- The client tier then displays to the user the data retrieved from the middle tier.
- The client tier never directly interacts with the information tier.

6 **Discuss the new features of WPF controls**



1. New WPF Controls:

i. DataGrid **ii. Calendar** **iii. DatePicker**

There are **three new** controls:

Let's look the brief description for these:

i. DataGrid controls:

A DataGrid is a control that displays data in a customizable grid. It provides a flexible way to display a collection of data in rows and columns.

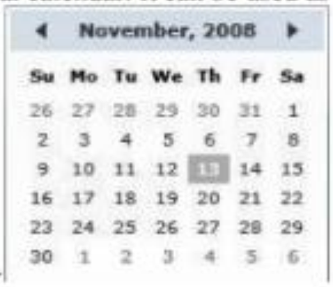
- It generates columns automatically, on the basis of the type of the data stored in them, by setting the

ItemSource property.

- Can perform various operations, such as grouping, sorting, and filtering on the data stored in the DataGrid control.
- Enables to perform validation on the **cell** and **row** levels.
  - Cell – validate the properties of a bound data object when another user updates the value.
  - Row – validate the entire data object when another user modifies the data in a row.

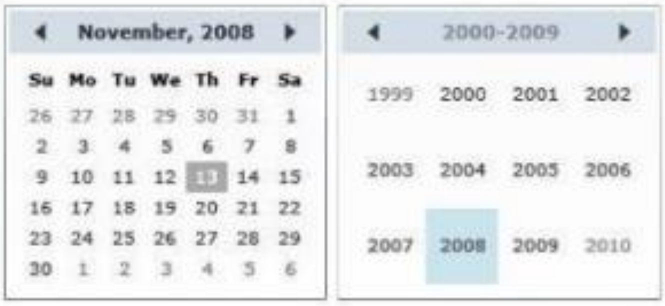
ii. Calendar Controls:

Calendar is a control that enables you to select a date from a visual calendar. It can be used as a separate control of the drop down part of the DatePicker control.

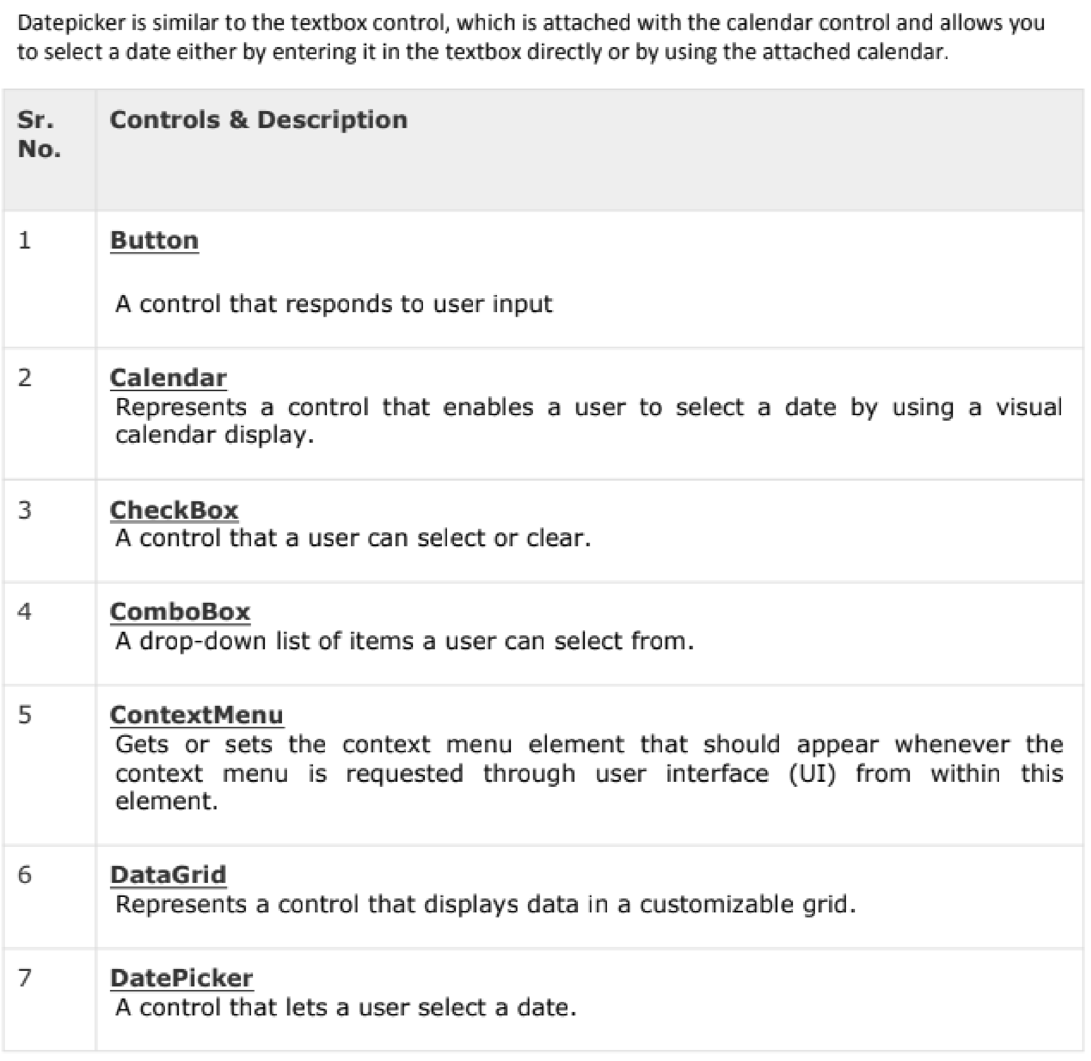


The default view of the Calendar control looks

We can change the Display mode to Year and Decade generates by setting the property **DisplayModes**.

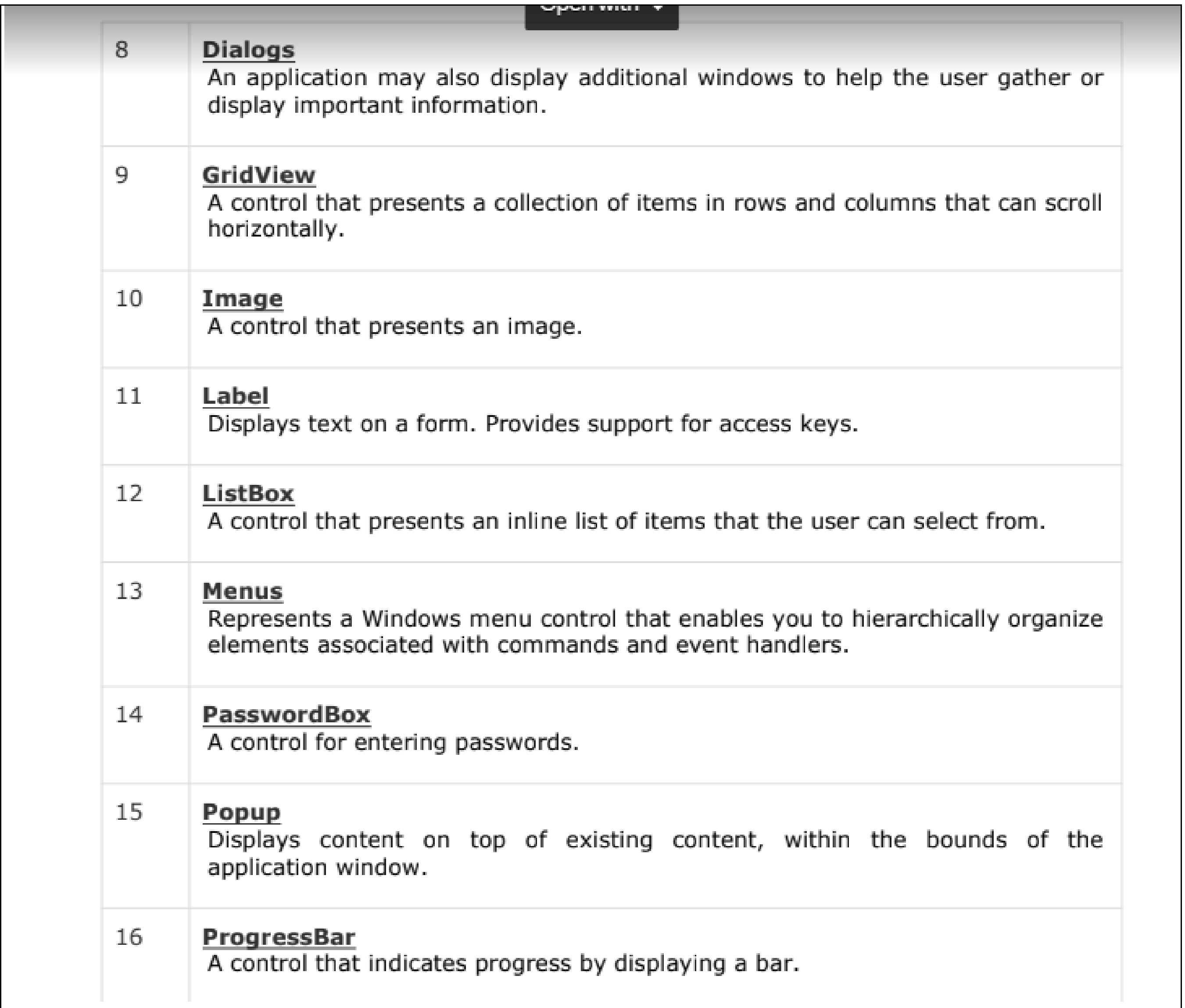


Datepicker:

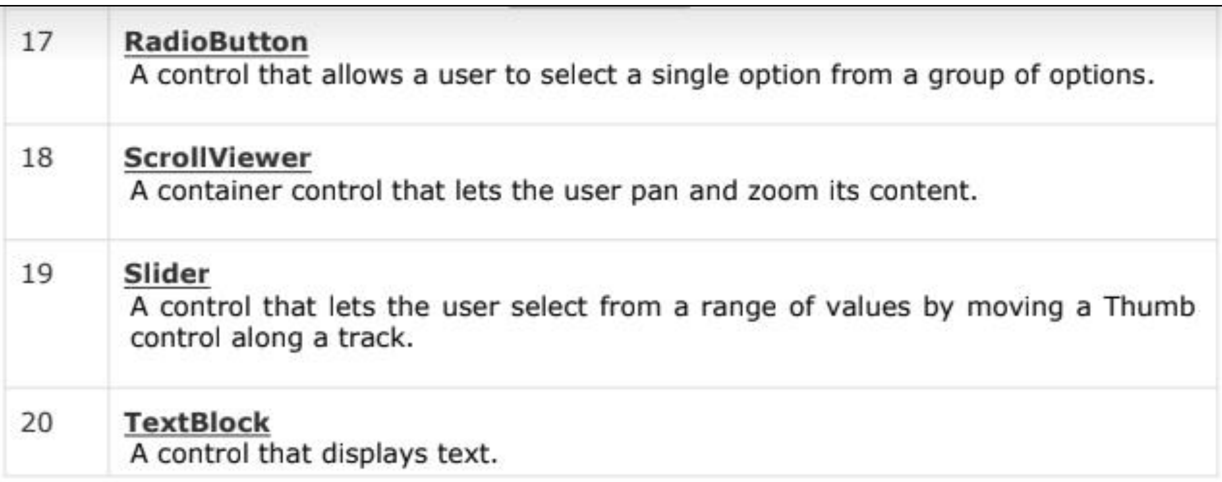


Datepicker is similar to the textbox control, which is attached with the calendar control and allows you to select a date either by entering it in the textbox directly or by using the attached calendar.

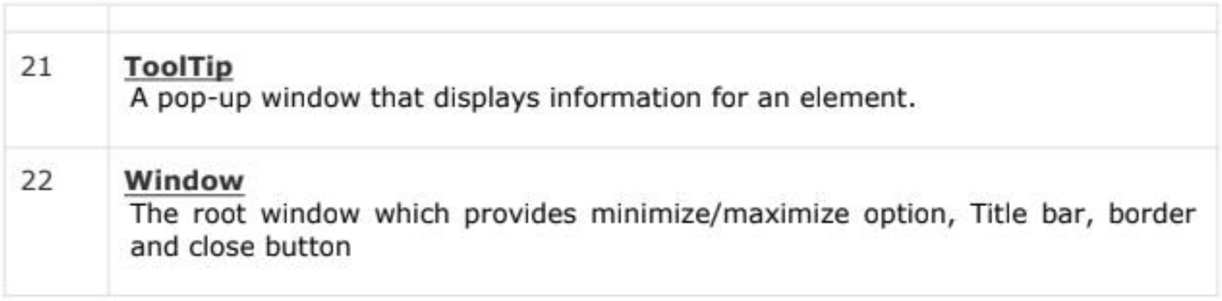
Sr. No.	Controls & Description
1	**Button** A control that responds to user input
2	**Calendar** Represents a control that enables a user to select a date by using a visual calendar display.
3	**CheckBox** A control that a user can select or clear.
4	**ComboBox** A drop-down list of items a user can select from.
5	**ContextMenu** Gets or sets the context menu element that should appear whenever the context menu is requested through user interface (UI) from within this element.
6	**DataGrid** Represents a control that displays data in a customizable grid.
7	**DatePicker** A control that lets a user select a date.



8	**Dialogs** An application may also display additional windows to help the user gather or display important information.
9	**GridView** A control that presents a collection of items in rows and columns that can scroll horizontally.
10	**Image** A control that presents an image.
11	**Label** Displays text on a form. Provides support for access keys.
12	**ListBox** A control that presents an inline list of items that the user can select from.
13	**Menus** Represents a Windows menu control that enables you to hierarchically organize elements associated with commands and event handlers.
14	**PasswordBox** A control for entering passwords.
15	**Popup** Displays content on top of existing content, within the bounds of the application window.
16	**ProgressBar** A control that indicates progress by displaying a bar.



17	**RadioButton** A control that allows a user to select a single option from a group of options.
18	**ScrollViewer** A container control that lets the user pan and zoom its content.
19	**Slider** A control that lets the user select from a range of values by moving a Thumb control along a track.
20	**TextBlock** A control that displays text.



21	**ToolTip** A pop-up window that displays information for an element.
22	**Window** The root window which provides minimize/maximize option, Title bar, border and close button

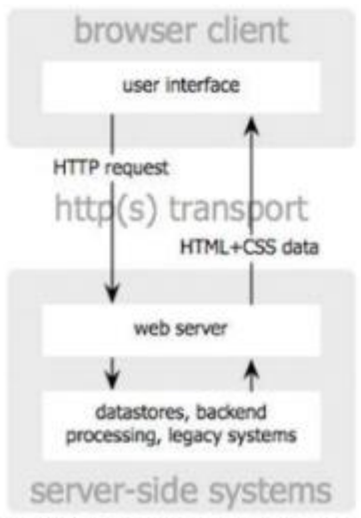
PART IV

7 **Explain in detailed about AJAX.**

4. AJAX and other Technologies:

Open with

AJAX is differ from other technologies, such as ASP.NET and JSP, which use the postback model. To understanding, you first need to analyse how other technologies work. Let's understanding the working of Traditional web applications with the below figure:



Traditional web application model

Traditional Web Applications are based on the request-response or postback model.

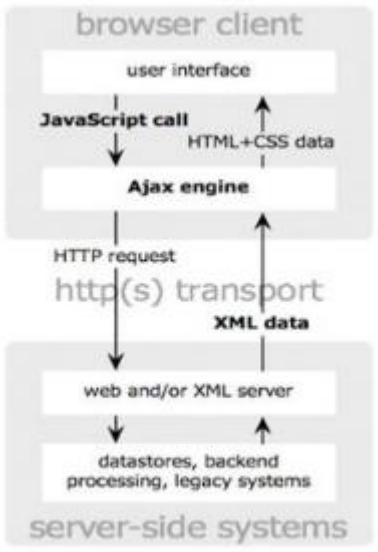
- ❖ Each client interaction with the UI of a traditional Web application, an Http request is sent to the server, which processes the request and sends the results back to the browser.
- ❖ At the time, when this request-response process is in progress, seeing a blank screen can be quite frustrating at times.
- ❖ Considering modern scenario, when more and more transactions are taking place through the Internet from all over the world, this time-consuming process can be frustrating.

To remove this shortcoming, AJAX, which implements an entirely different approach to web development, has been introduce as shows the AJAX Web Application model:

The AJAX Web Application model handles the client-side activities, such as validations, without making a round trip to the server.

The tasks that are required to be processed at the server-end are handled by the AJAX engine.

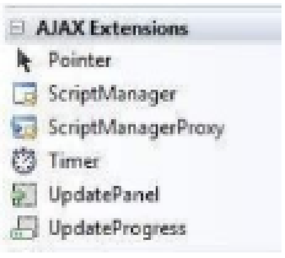
The AJAX engine makes **asynchronous** calls to the server without interrupting the user's interactions with the UI; therefore it enhances the user-interactivity and performance, which makes AJAX different from other technologies.



AJAX web application model

5. AJAX Control ToolKit:

The control toolbox in the Visual Studio IDE contains a group of controls called the "AJAX Extensions".



The ScriptManager Control:

The ScriptManager control is the most important control and must be present on the page for other controls to work.

Syntax:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control.

The **ScriptManager** control takes care of the client-side script for all the server side controls.

The UpdatePanel Control:

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface.

When a control inside it triggers a post back, the UpdatePanel interferes to initiate the post asynchronously and update just that portion of the page.

Example:

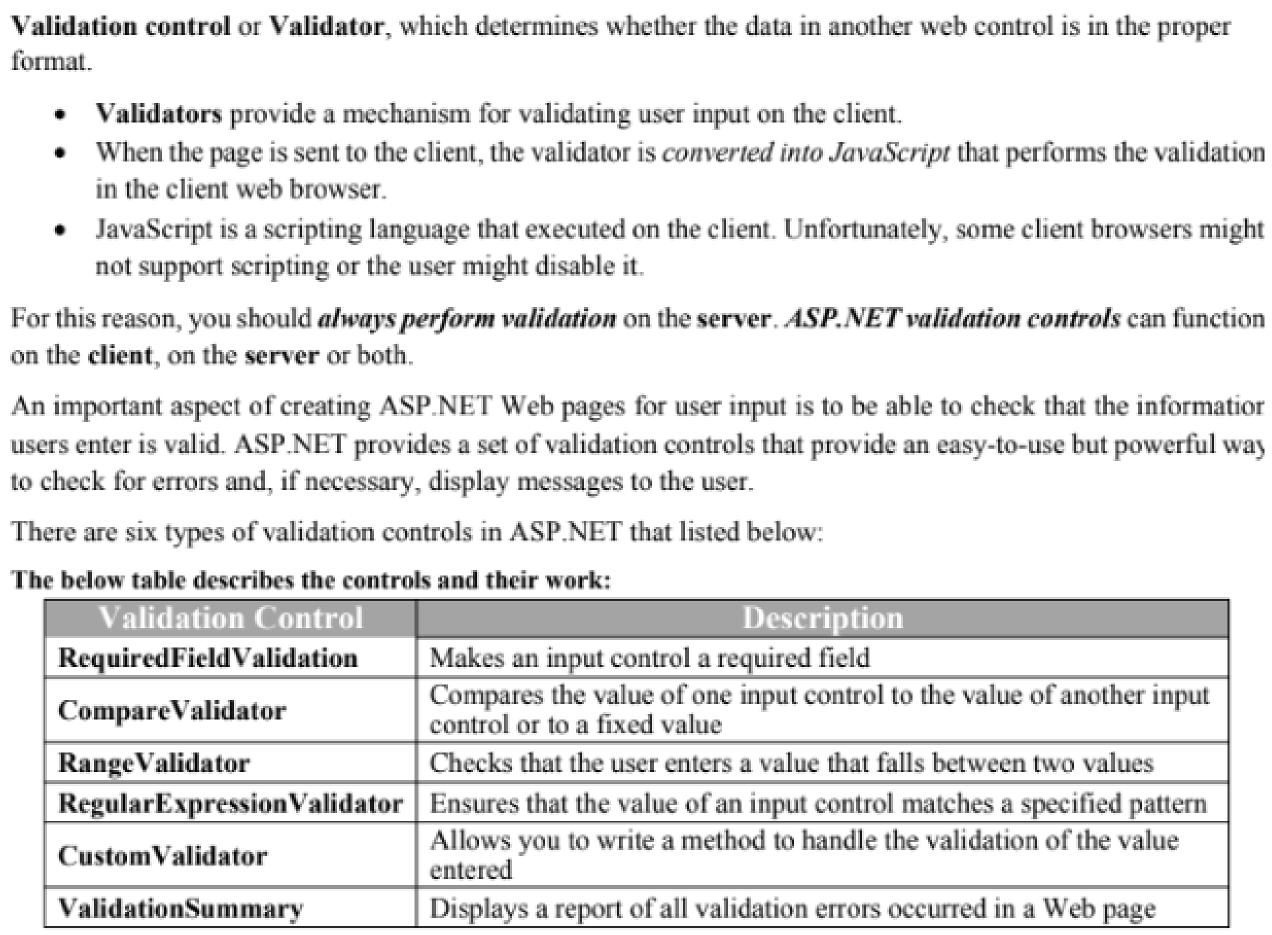
```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
                <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click"
                        Text="Partial PostBack"/>
                <br />
                <br />
                <asp:Label ID="lblpartial" runat="server"></asp:Label>
        </ContentTemplate>
</asp:UpdatePanel>
```

Properties of the UpdatePanel Control

The following table shows the properties of the update panel control:

Properties	Description
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

8 **Explain different validation controls with suitable example**



Validation control or **Validator**, which determines whether the data in another web control is in the proper format.

- **Validators** provide a mechanism for validating user input on the client.
- When the page is sent to the client, the validator is *converted into JavaScript* that performs the validation in the client web browser.
- JavaScript is a scripting language that executed on the client. Unfortunately, some client browsers might not support scripting or the user might disable it.

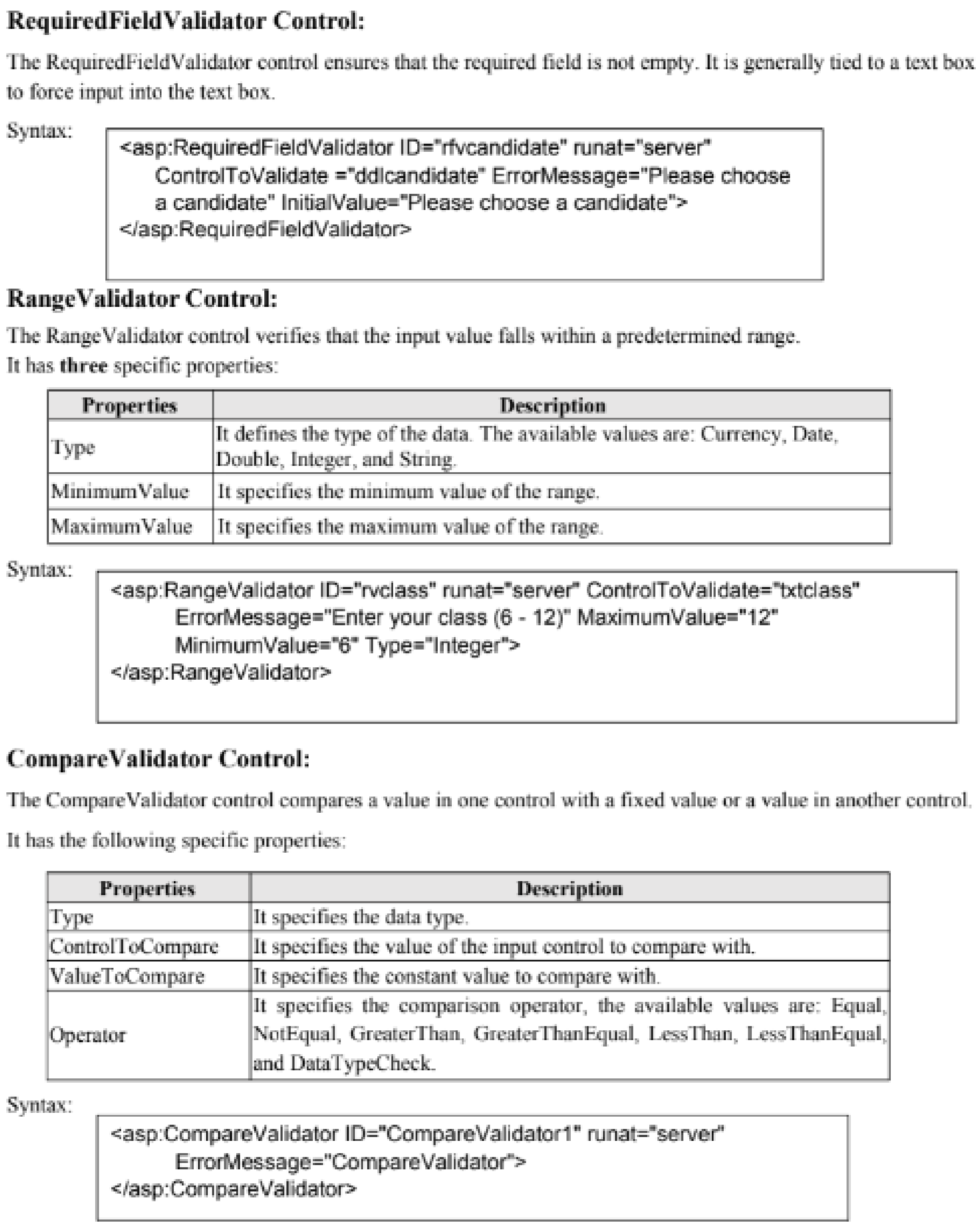
For this reason, you should ***always perform validation*** on the **server**. ***ASP.NET validation controls*** can function on the **client**, on the **server** or both.

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the informatior users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET that listed below:

The below table describes the controls and their work:

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
CustomValidator	Allows you to write a method to handle the validation of the value entered
ValidationSummary	Displays a report of all validation errors occurred in a Web page



RequiredFieldValidator Control:

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

Syntax:

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

RangeValidator Control:

The RangeValidator control verifies that the input value falls within a predetermined range.
It has **three** specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

Syntax:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
        ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
        MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

CompareValidator Control:

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

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

Syntax:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
        ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

RegularExpressionValidator:

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.

The following table summarizes the commonly used syntax constructs for

Regular expressions: \b, \n, \,\f, \r, \v, \t

Metacharacters: ., [abcd], [^abcd], [a-zA-Z], \w, \W, \s, \S, \d, \D

Quantifiers: *, +, ?, {n}, {n, }, {n,m}

Syntax:

```
<asp:RegularExpressionValidator ID="string" runat="server"
        ErrorMessage="string" ValidationExpression="string"
        ValidationGroup="string">
</asp:RegularExpressionValidator>
```

CustomValidator:

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** eventhandler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

Syntax:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
        ClientValidationFunction=.cvf_func.
        ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

ValidationSummary:

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

Syntax:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
        DisplayMode = "BulletList" ShowSummary = "true"
        HeaderText="Errors:"  />
```

Validation Groups:

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using **validation groups.**

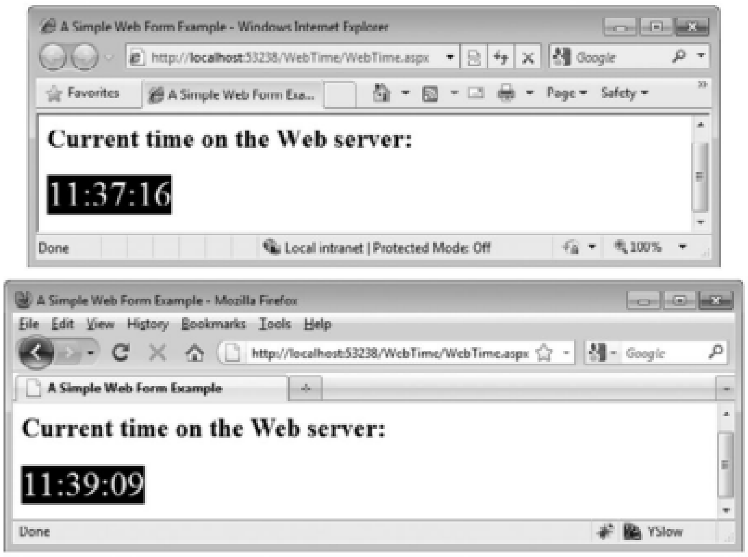
To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their **ValidationGroup** property.

PARTV

9 **How to create a website using ASP.Net**

Your First Web Application

Our first example displays the web server's time of day in a browser window (Fig. 19.4). When this application executes—that is, a web browser requests the application's web page—the web server executes the application's code, which gets the current time and dis-plays it in a `Label`. The web server then returns the result to the web browser that made the request, and the web browser renders the web page containing the time. We executed this application in both the Internet Explorer and Firefox web browsers to show you that the web page renders identically in each.

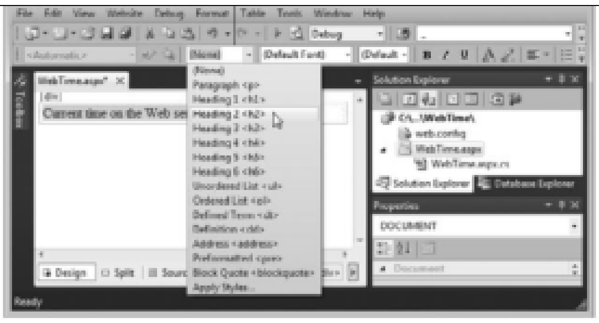

A Simple Web Form Example - Windows Internet Explorer
http://localhost:53238/WebTime/WebTime.aspx
Favorites
A Simple Web Form Exa...
Done
Local intranet | Protected Mode: Off
100%
A Simple Web Form Example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:53238/WebTime/WebTime.aspx
A Simple Web Form Example
Done
YSlow

Current time on the Web server:

11:37:16

Current time on the Web server:

11:39:09



tem is
ct the

efault

Click to the right of the text you just typed and press the Enter key to start a new paragraph in the page

19.4.2 Examining WebTime.aspx's Code-Behind File

Figure 19.20 presents the code-behind file WebTime.aspx.cs. Line 5 begins the declara-tion of class WebTime. In Visual C#, a class declaration can span multiple source-code files—the separate portions of the class declaration in each file are known as **partial classes**. The partial **modifier** indicates that the code-behind file is part of a larger class. Like Win-dows Forms applications, the rest of the class's code is generated for you based on your vi-sual interactions to create the application's GUI in Design mode. That code is stored in other source code files as partial classes with the same name. The compiler assembles all the partial classes that have the same into a single class declaration.

e that
u can

Line 5 indicates that WebTime inherits from class **Page** in namespace

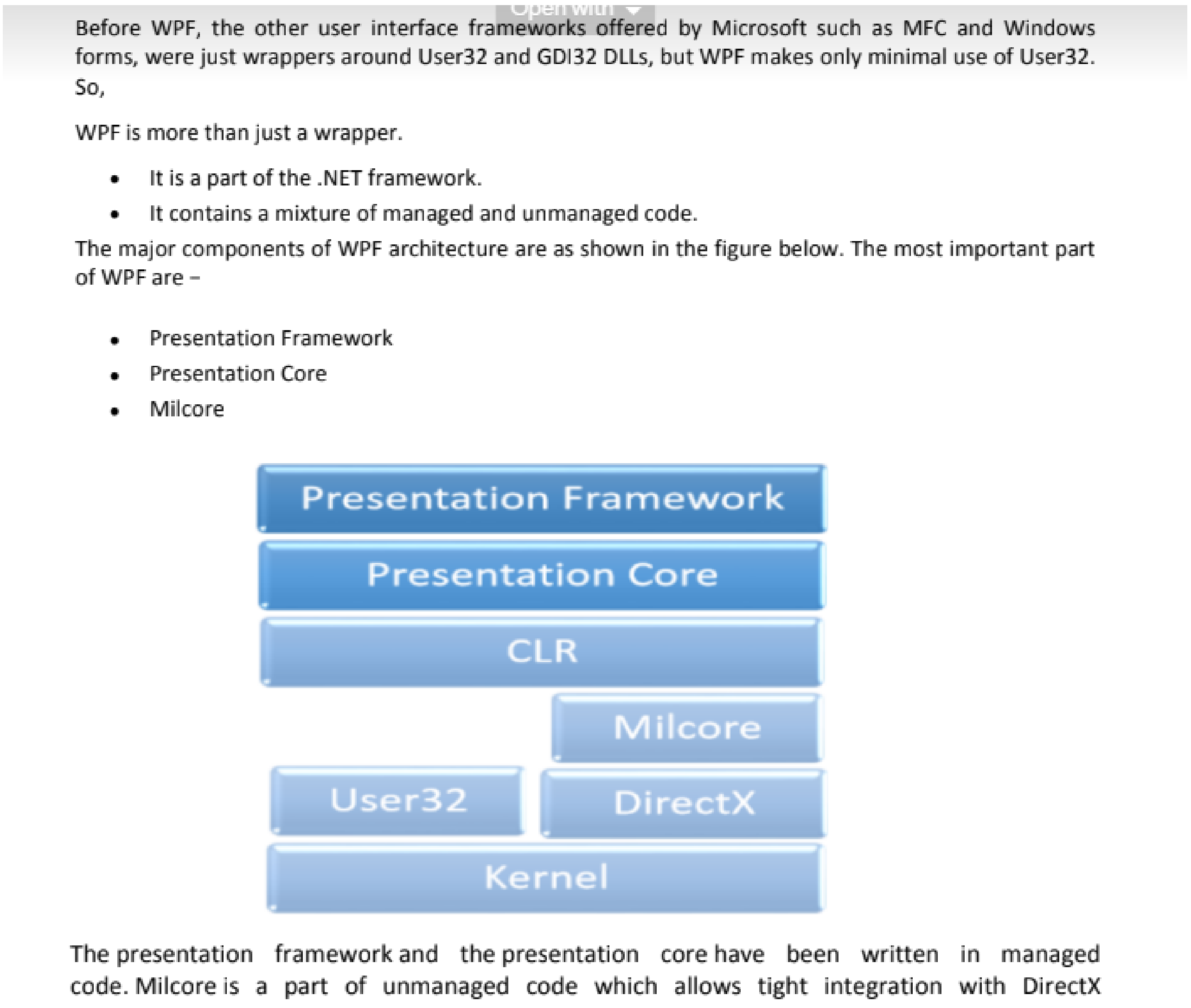
System.Web.UI. This namespace contains classes and controls for building web-based applications. Class Page represents the default capabilities of each page in a web application—all pages inherit directly or indirectly from this class.

Lines 8–12 define the Page_Init event handler, which initializes the page in response to the page's Init event. The only initialization required for this page is to set the time-Label's Text property to the time on the web server computer. The

statement in line 11 retrieves the current time (DateTime.Now) and formats it as *hh:mm:ss*. For example, 9 AM is formatted as 09:00:00, and 2:30 PM is formatted as 02:30:00. As you'll see, variable timeLabel represents an ASP.NET Label control. The ASP.NET controls are defined in namespace **System.Web.UI.WebControls**.

```
    // Fig. 19.20: WebTime.aspx.cs
    // Code-behind file for a page that displays the web server's time.
3using System;
4
5public partial class WebTime : System.Web.UI.Page
6{
7// initializes the contents of the page
8protected void Page_Init( object sender, EventArgs e )
        {
            // display the server's current time in timeLabel
            timeLabel.Text =  DateTime.Now.ToString( "hh:mm:ss" );
        } // end method Page_Init
    } // end class WebTime
```

10 **Explain in detailed about WPF architecture**



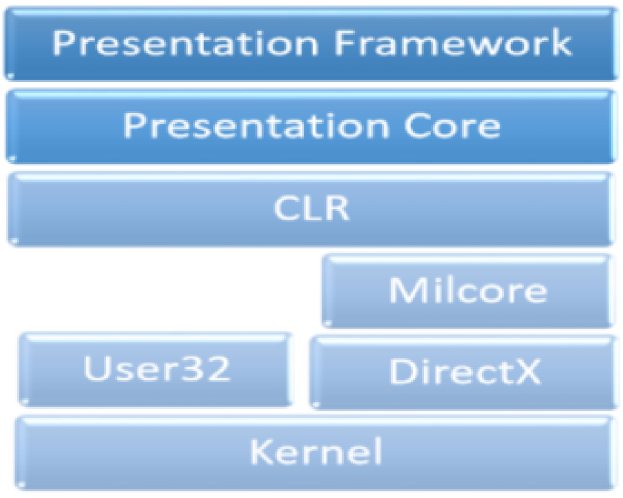
Before WPF, the other user interface frameworks offered by Microsoft such as MFC and Windows forms, were just wrappers around User32 and GDI32 DLLs, but WPF makes only minimal use of User32. So,

WPF is more than just a wrapper.

- It is a part of the .NET framework.
- It contains a mixture of managed and unmanaged code.

The major components of WPF architecture are as shown in the figure below. The most important part of WPF are –

- Presentation Framework
- Presentation Core
- Milcore



The presentation framework and the presentation core have been written in managed code. Milcore is a part of unmanaged code which allows tight integration with DirectX (responsible for display and rendering). CLR makes the development process more productive by offering many features such as memory management, error handling, etc.

The architecture of WPF is actually a multilayered architecture. It has mainly three layers, the WPF Managed Layer, the WPF Unmanaged Layer and the Core operating system element, basically these layers are a set of assemblies that rovide the entire framework.

The Presentation Framework, Presentation Core and Window Base are the three major components of the Managed Layer. These are the major code portions of WPF and plays a vital role in an overview of Windows Presentation Foundation. The public API exposed is only via this layer. The Major portion of the WPF is in managed code.

- PresentationFramework.dll: This section contains high-level features like application windows, panels, styles controls, layouts, content and so on that helps us to build our application. It also implements the end-user presentation features including data binding, time-dependencies, animations and many more.

- PresentationCore.dll: This is a low-level API exposed by WPF providing features for 2D, 3D, geometry and so on. The Presentation Core provides a managed wrapper for MIL and implements the core services for WPF such as UI Element and visual . The Visual System creates visual tree that contains applications Visual Elements and rendering instructions.

- WindowsBase.dll: It holds the more basic elements that are capable to be reused outside the WPF environment like Dispatcher objects and Dependency objects.

UnManaged Layer

- milCore.dll: The composition engine that renders the WPF application is a native component. It is called the Media Integration Layer (MIL) and resides in milCore.dll. The purpose of the milCore is to interface directly with DirectX and provide basic support for 2D and 3D surface. This section is unmanaged code because it acts as a bridge between WPF managed and the DirectX / User32 unmanaged API.

- WindowsCodecs.dll: WindowsCodecs is another low-level API for imaging support in WPF applications like image processing, image displaying and scaling and so on. It consists of a number of codecs that encode/decode images into vector graphics that would be rendered into a WPF screen.

Core operating System Layer (Kernel)

This layer has OS core components like User32, GDI, Device Drivers, Graphic cards and so on. These components are used by the application to access low-level APIs.

- DirectX: DirectX is the low-level API through which WPF renders all graphics. DirectX talks with drivers and renders the content.

- User32: User32 actually manages memory and process separation. It is the primary core API that every application uses. User32 decides which element will be placed where on the screen.

- GDI: GDI stands for Graphic Device Interface. GDI provides an expanded set of graphics primitives and a number of improvements in rendering quality.

- CLR: WPF leverages the full .NET Framework and executes on the Common Language Runtime (CLR).