

Building Windows Forms Application

Objectives

- ◆ Overview Windows Forms (WinForm)
- ◆ Explain about type Controls and Components in WinForm
- ◆ Working with Common Controls: Label, TextBox, MaskedTextBox, etc
- ◆ Working with Menu, ProgressBar and Timer in WinForm
- ◆ Explain about Position and Layout in WinForm
- ◆ Demo create Windows Forms Application
- ◆ Demo creating and using Menu in WinForm
- ◆ Demo create MDI Application in WinForm

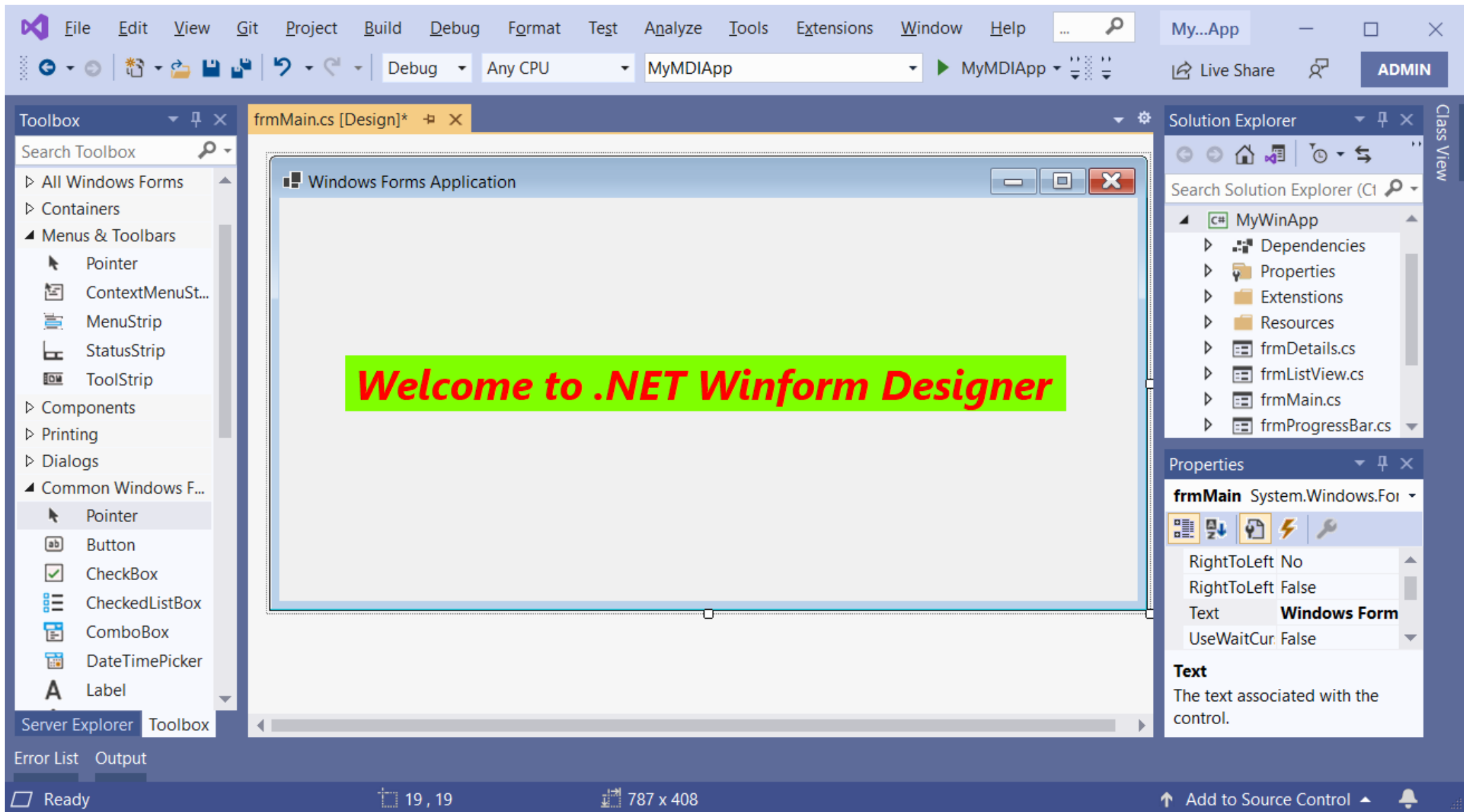
Overview Windows Forms

Introduction to Windows Forms

- ◆ Windows Forms is a UI framework for building desktop apps. It provides one of the most productive ways to create desktop apps based on the visual designer provided in Visual Studio
- ◆ With Windows Forms, we develop graphically rich apps that are easy to deploy, update, and work while offline or while connected to the internet
- ◆ Windows Forms apps can access the local hardware and file system of the computer where the app is running
- ◆ Windows Forms has rich UI controls that emulate features in high-end apps like Microsoft Office
- ◆ Functionality such as drag-and-drop placement of visual controls makes it easy to build desktop apps

Introduction to Windows Forms

- ◆ The Windows Forms development platform supports a broad set of app development features, including controls, graphics, data binding, and user input
- ◆ Windows Forms features a drag-and-drop visual designer in Visual Studio to easily create Windows Forms apps
- ◆ Windows Forms is a UI technology for .NET, a set of managed libraries that simplify common app tasks such as reading and writing to the file system
- ◆ When we use a development environment like Visual Studio, we can create Windows Forms smart-client apps that display information, request input from users, and communicate with remote computers over a network



Introduction to Windows Forms

- ◆ Click one deployment
- ◆ Application setting
- ◆ New Windows Forms controls
- ◆ New Data Binding model
- ◆ Rich graphics

Create App with least operation

Store global data to reuse anywhere

ToolStrip, MenuStrip, ProgressBar, etc.

Powerfull way to link control with data source (BindingSource component)

GDI+ : drawing and painting image on forms

Create WinForm Application Demonstration

1. Open Visual Studio
2. Select **Create a new project**

Visual Studio 2019


Open recent

Search recent (Alt+S) 🔍


Yesterday

 Slot_12_File_Stream_IO.sln 2/26/2021 7:43 AM
D:\Demo\FU\Basic.NET\Slot_12_File_StreamI_O

This week

 FunWithProbingPaths.sln 2/25/2021 8:27 AM
D:\Demo\FU\FunWithProbingPaths

 FunWithProbingPaths.sln 2/25/2021 8:24 AM
D:\...\Demo\pro-c-sharp-8-w-.net-core-3-master\Chapte...

 Buffers_Console_Client.csproj 2/25/2021 8:16 AM
C:\Users\MICHA~1\AppData\Local\Temp\Rar\$DIa1445...

 Buffers_Console_Client.csproj 2/25/2021 8:15 AM
D:\...\pro-.net5-custom-libraries-main\Ch03\RVJ.Core\Buf...

Get started



Clone a repository

Get code from an online repository like GitHub or Azure DevOps



Open a project or solution

Open a local Visual Studio project or .sln file



Open a local folder

Navigate and edit code within any folder

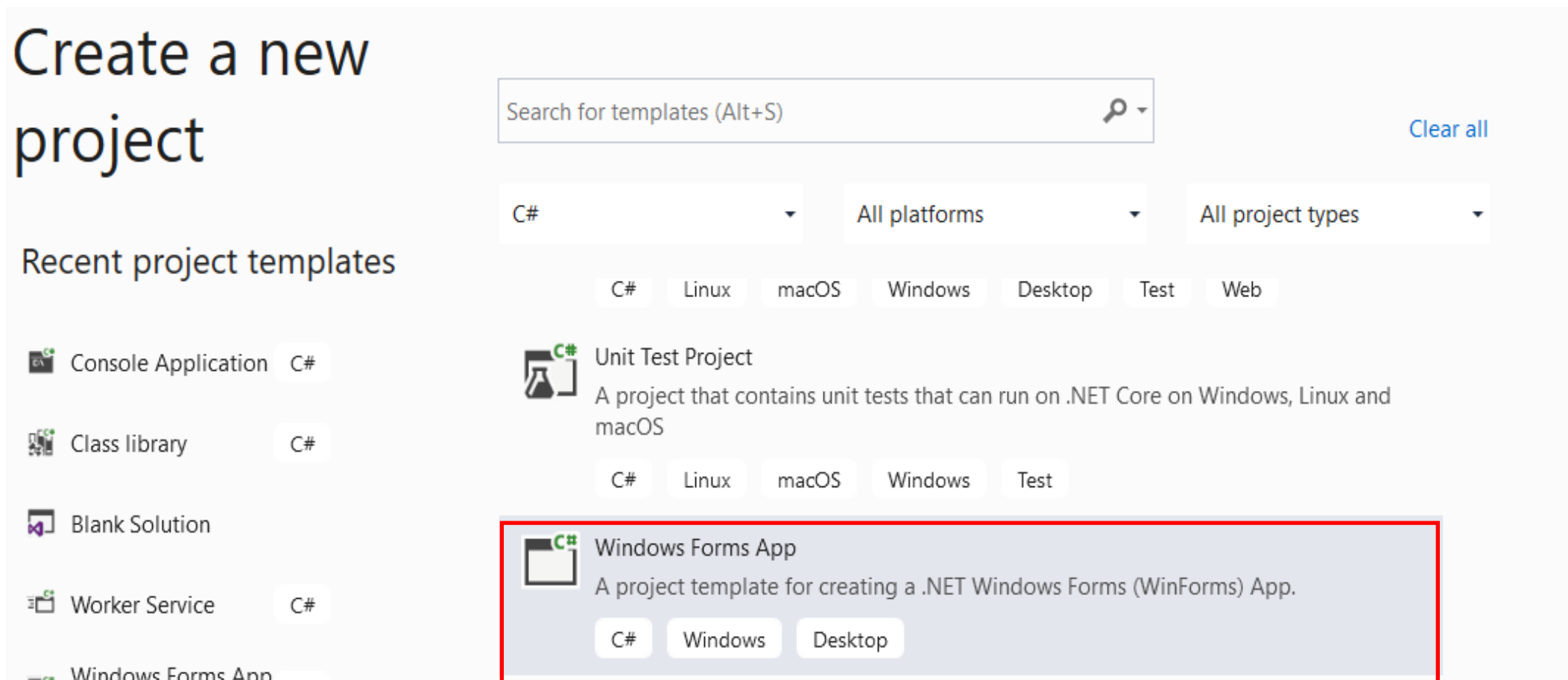


Create a new project

Choose a project template with code scaffolding to get started

[Continue without code →](#)

3. In the **Search for templates** box, type **winforms**, and then press **Enter**
4. In the code language dropdown, choose **C#**
5. In the templates list, select Windows Forms App(.NET) and then click **Next**



6. In the **Configure your new project** window, set the **Project name** to **MyWinApp** and click **Next** (save project to a folder by adjusting the **Location** setting)
7. In the **Additional information** window, select **Target Framework** to **.NET 5.0** and click **Create**

Configure your new project

Windows Forms App C# Windows Desktop

Project name

MyWinApp

Location

D:\Demo\FU\Basic.NET\Slot_13_14_WindowsForms\

Solution name ⓘ

MyWinApp

☐ Place solution and project in the same directory

Back Next

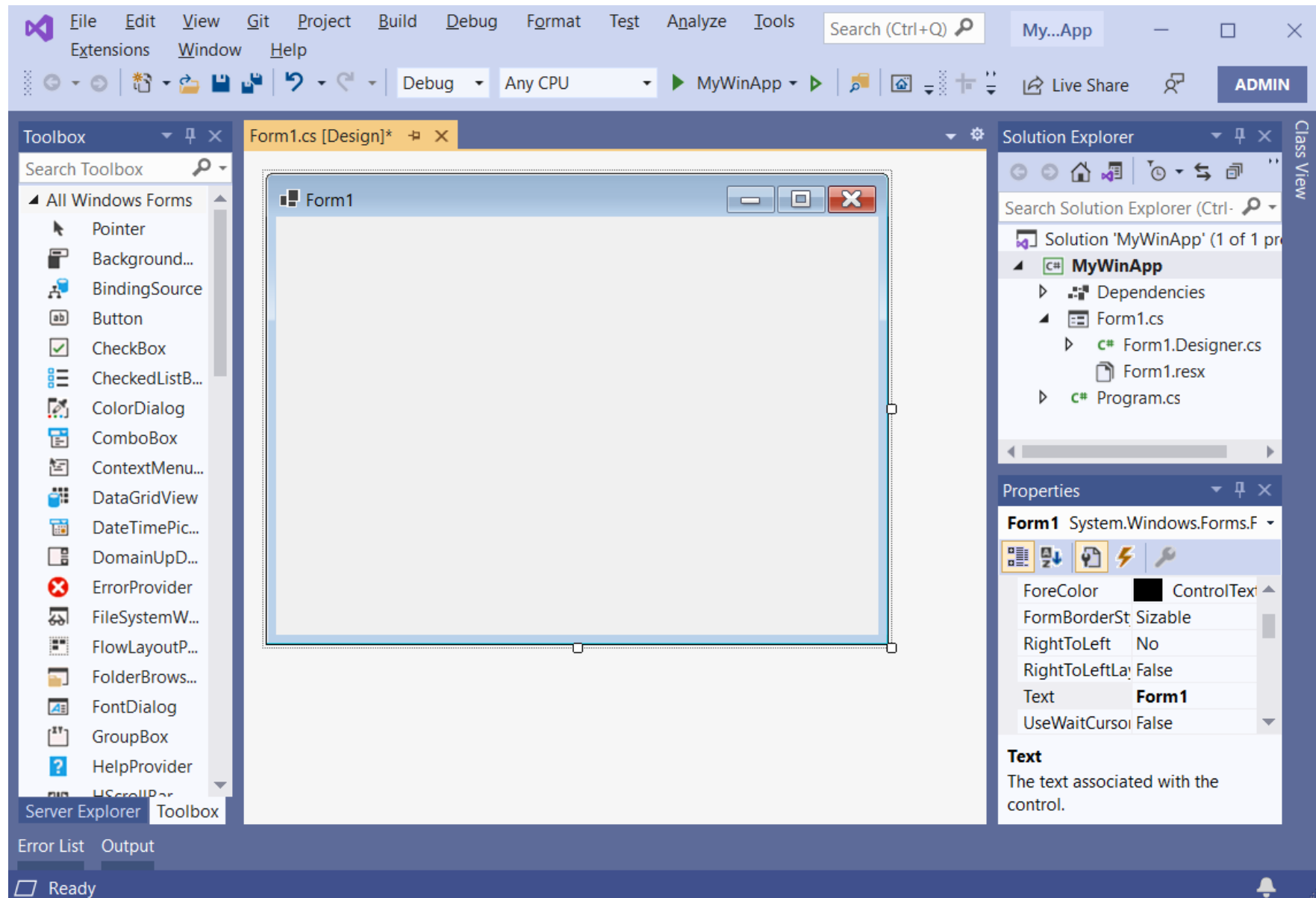
Additional information

Windows Forms App C# Windows Desktop

Target Framework

.NET 5.0 (Current)

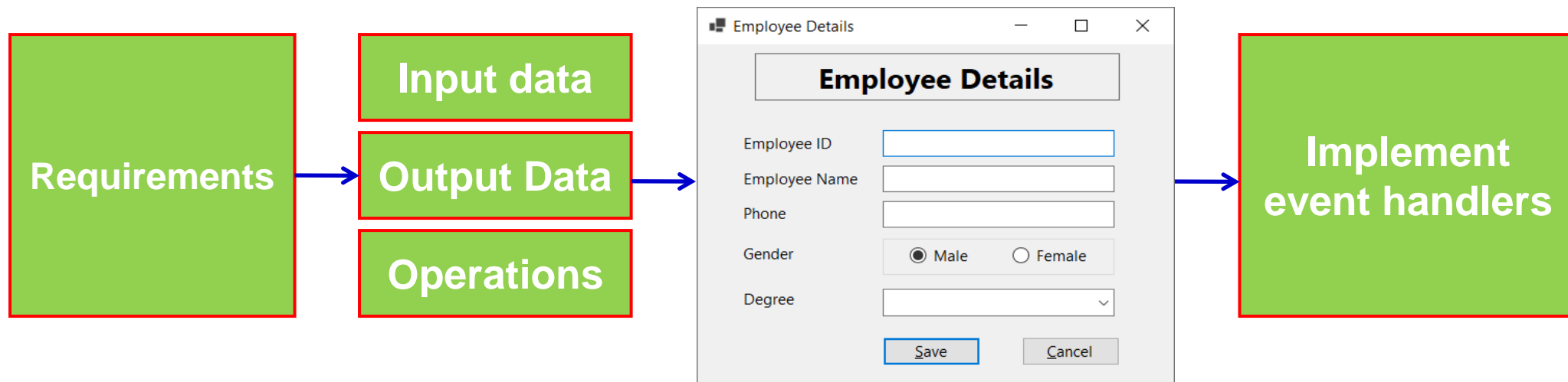
Back Create



Working with Controls in WinForm

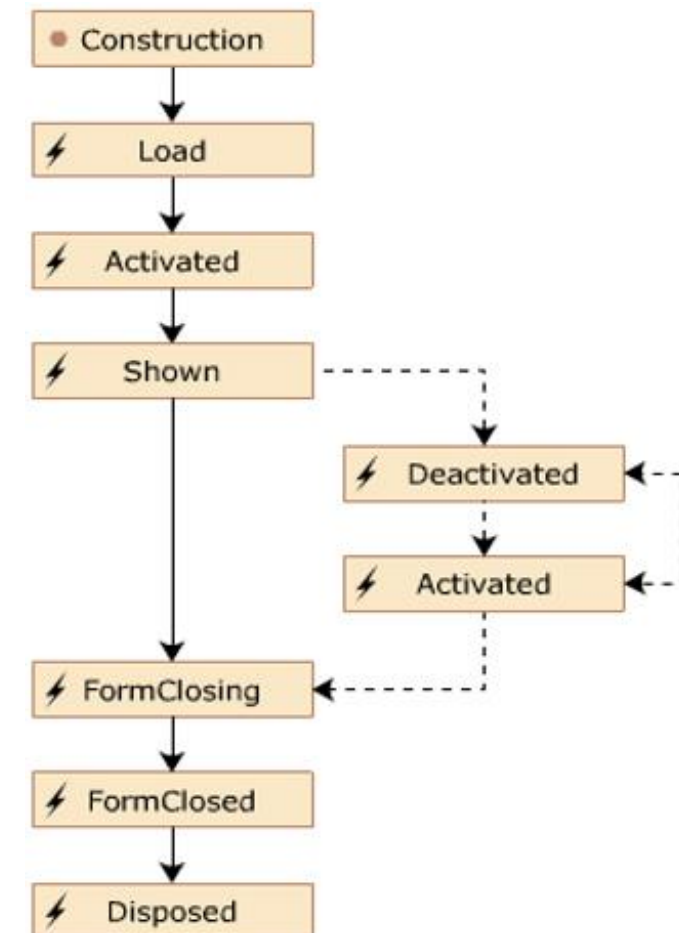
A Strategy for Designing the GUI

- ◆ Identify needed controls / components
- ◆ Isolate regions and behaviors
- ◆ Sketch (phác họa) the GUI
- ◆ Choose Layout managers



Forms Class

- ◆ Is represented as a class in the System.Windows.Forms namespace
- ◆ Is the basic unit of an application
- ◆ Presents information to the user
- ◆ Receives information from the user



Properties	Text, BackColor, ForeColor, Font, WindowState, AcceptButton, Location, Modal, Name
Methods	Activate, Close, Focus, Hide, Show, Enabled
Events	Load, Activated, Shown, Deactivate, FormClosing, FormClosed, Resize, Click, GotFocus

Types of Control

- ◆ **Base Control Class:** The Control class is the base class for Windows Forms controls. It provides the infrastructure required for visual display in Windows Forms applications and provides the following capabilities:
 - Exposes a window handle. Manages message routing
 - Provides mouse and keyboard events, and many other user interface events
 - Provides advanced layout features
- ◆ **Composite Controls:** A composite control is a collection of Windows Forms controls encapsulated in a common container. This kind of control is sometimes called a **user control**

Types of Control

- ◆ **Extended Controls:** It is a control that is created by inheriting from any existing Windows Forms control. With this approach, we can keep all of the inherent functionality of a Windows Forms control, and then extend that functionality by adding custom properties, methods, or other features
- ◆ **Custom Controls:** It is a control that is created by inheriting from **Control** class. The **Control** class provides all of the basic functionality required by controls, including mouse and keyboard handling events, but no control-specific functionality or graphical interface

Control Class

- ◆ **Control class:** Base class of all controls available in WinForms

Properties	Method	Event
<ul style="list-style-type: none"> • CanFocus • Controls • Enable • Name • Parent • TabIndex • Visible 	<ul style="list-style-type: none"> • Focus • GetNextControl • Hide • IsMnemonic • Select • Show 	<ul style="list-style-type: none"> • Click • ControlAdd • DoubleClick • Validating • Validated • KeyPress • Leave • LostFocus • MouseClick • Move

Label

- Label controls are used to display text that cannot be edited by the user. They're used to identify objects on a form and to provide a description of what a certain control represents or does

Properties

- Name
- Text
- TextAlign
- UseMnemonic

Method

- Contains
- Hide
- Show

Event

- Click

```
Label fName = new Label();
fName.Text = "&First Name : ";
fName.UseMnemonic = true ;
```

TextBox

- ◆ A TextBox control accepts user input on a Form

Properties

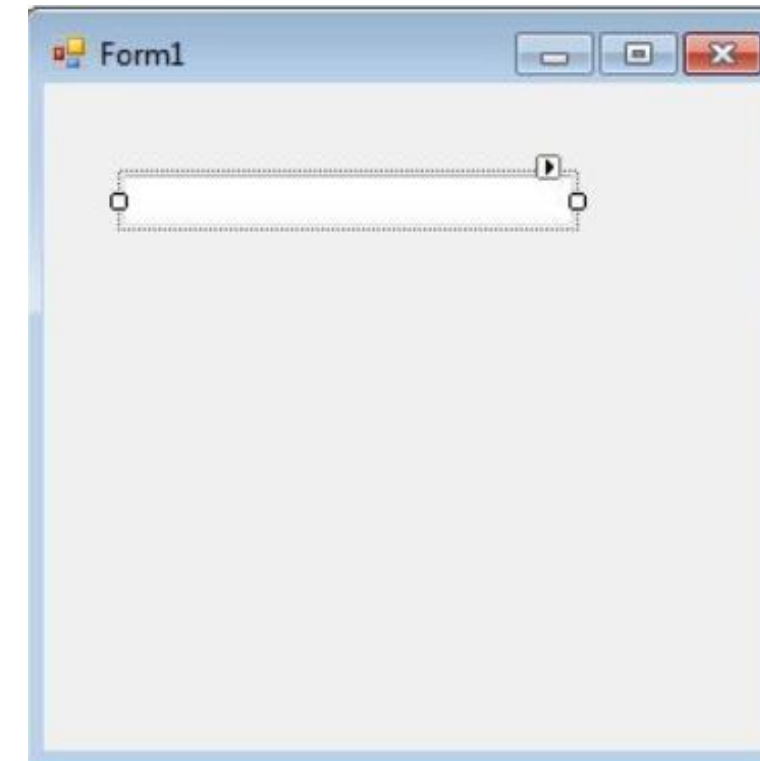
- CharacterCasing
- MaxLength
- MultiLine
- Name
- PasswordChar
- ReadOnly
- Text

Method

- AppendText
- Clear
- Focus
- Copy
- Paste

Event

- KeyPress
- Leave
- TextChanged



MaskedTextBox

- ◆ A MaskedTextBox control provides a validation mechanism for user input on a Form. When we want a TextBox to accept a date in mm/dd/yyyy format, we can set masking in the MaskedTextBox

Properties

- Mask
- MaskFull
- MaskCompleted
- Name
- PromptChar
- Text

Method

- SelectAll
- GetPositionFromCharIndex

Event

- MaskChanged
- MaskedInputRejected

Input Mask

Select a predefined mask description from the list below or select Custom to define a custom mask.

Mask Description	Data Format	Validating Type
Numeric (5-digits)	12345	Int32
Phone number	(574) 555-0123	(none)
Phone number no area co...	555-0123	(none)
Short date	12/11/2003	DateTime
Short date and time (US)	12/11/2003 11:20	DateTime
Social security number	000-00-1234	(none)
Time (European/Military)	23:20	DateTime
Time (US)	11:20	DateTime
Zip Code	98052-6399	(none)
<Custom>		(none)

Mask: 00/00/0000 ☒ Use ValidatingType

Preview: __/__/____

OK Cancel

MaskedTextBox

Mask	Description
0	Digit, required. [0-9]
9	Digit or space, optional
#	Digit or space, optional. If this position is blank in the mask, it will be rendered as a space in the Text property. Plus (+) and minus (-) are allowed
L	Letter, required [a-zA-Z]
?	Letter, optional. [a-zA-Z]
&	Character, required. If the AsciiOnly property = true, behaves like the "L"
C	Character, optional. Any non-control character. If the AsciiOnly property =true, this element behaves like the "?" element
A	Alphanumeric, required. If the AsciiOnly property =true, the only characters it will accept are the ASCII letters a-z and A-Z
a	Alphanumeric, optional. If the AsciiOnly property is set to true, the only characters it will accept are the ASCII letters a-z and A-Z

Button

- ◆ A Button control is a child control placed on a Form and used to process click event and can be clicked by a mouse click or by pressing ENTER or ESC keys

Properties

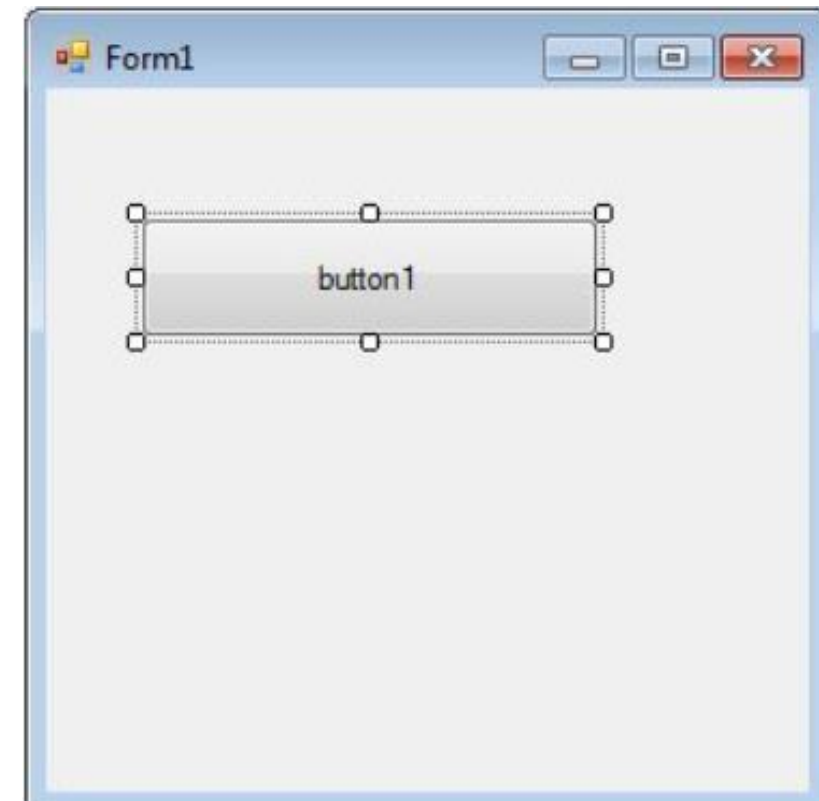
- DialogResult
- Enabled
- FlatStyle
- Image
- Name
- Text

Method

- Focus
- PerformClick

Event

- Click
- DoubleClick
- MouseDoubleClick



ListBox

- ◆ A ListBox control provides a user interface to display a list of items. Users can select one or more items from the list
- ◆ A ListBox may be used to display multiple columns and these columns may have images and other controls

Properties

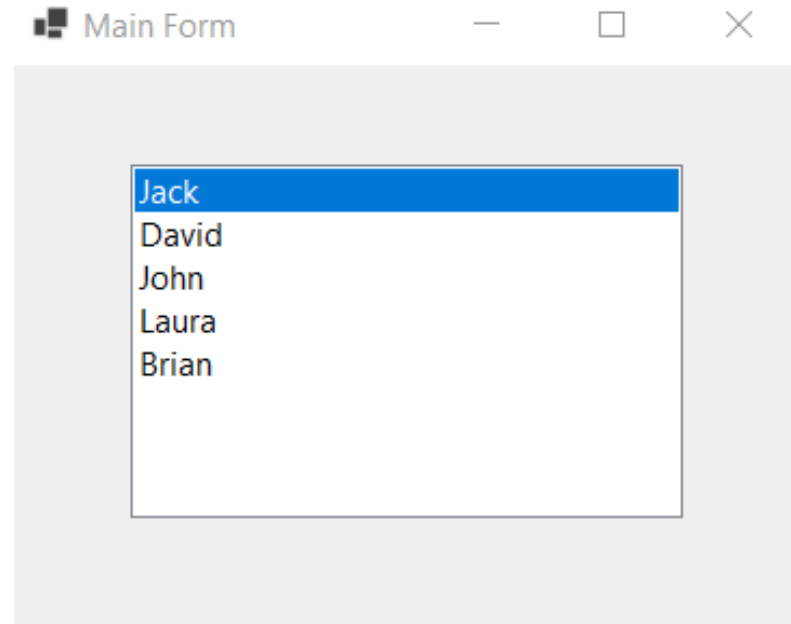
- DisplayMember
- Items
- SelectionMode
- SelectedIndex

Method

- ClearSelected
- GetItemText
- GetSelected
- SetSelected

Event

- SelectedIndexChanged
- SelectedValueChanged
- ValueMemberChanged



ComboBox

- ◆ The ComboBox control provides combined functionality of a text box and a listbox in a single control. Only one list item is displayed at one time in a ComboBox and rest of the available items are loaded in a drop down list

Properties

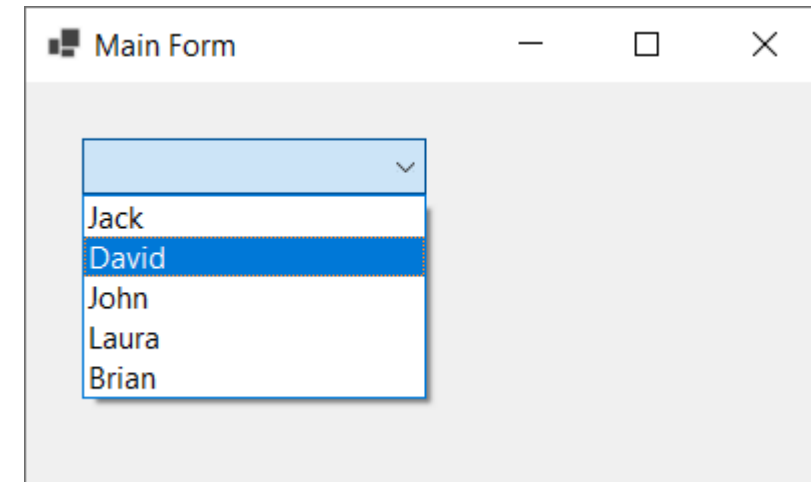
- DropDownStyle
- Items
- MaxDropDownItem
- SelectedItem
- SelectedIndex
- Text
- ValueMember

Method

- GetItemText
- SelectAll
- Select (int start, int len)

Event

- DropDown
- SelectedIndexChanged
- SelectedValueChanged
- ValueMemberChanged



More Controls

◆ Value Setting ControlForm class

- RadioButton
- CheckBox
- CheckListBox

◆ Grouping Controls

- GroupBox
- Panel

◆ Images Control

- PictureBox
- ImageList

2 Door
4 Door
Hatchback

4 Cylinder
6 Cylinder
8 Cylinder

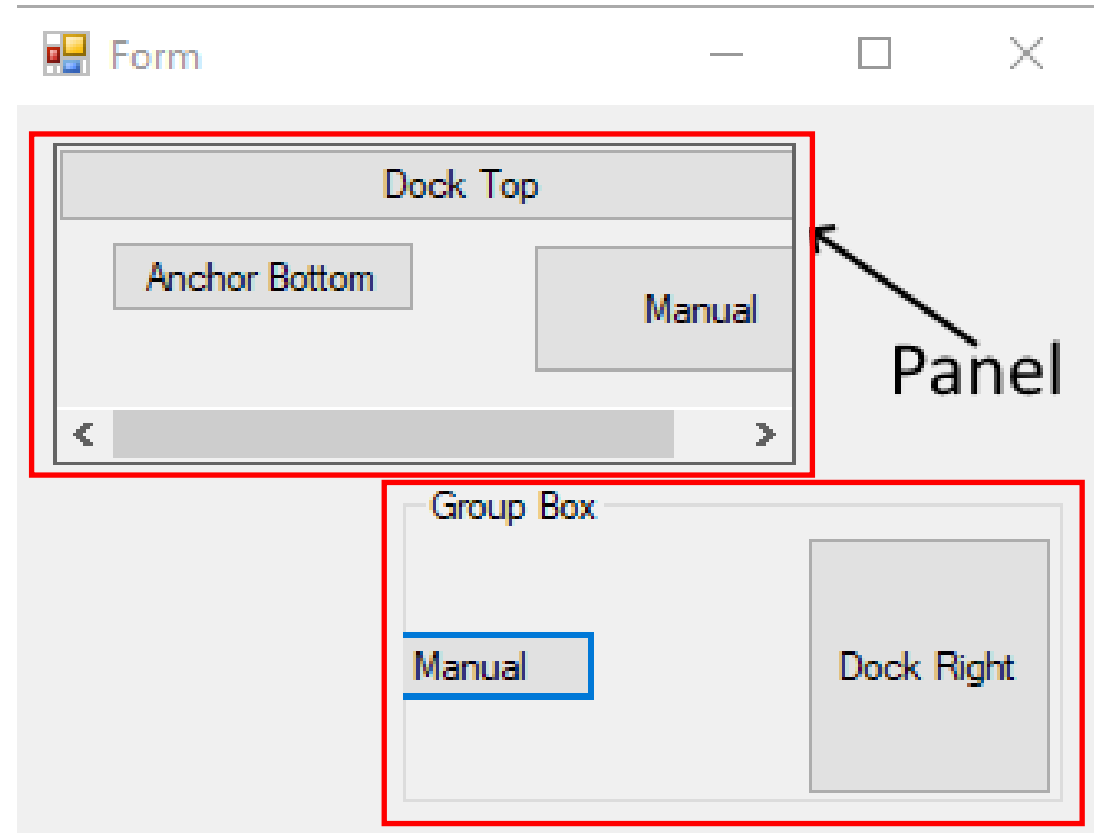
Options

☒ Air conditioning
☒ Cruise Control
☒ Power Steering

Submit

Grouping Control

- ◆ **GroupBox:** Represents a Windows control that displays a frame around a group of controls with an optional caption
- ◆ **Panel:** Used to group collections of controls.



RadioButton

- ◆ A RadioButton control provides a round interface to select one option from a number of options. Radio buttons are usually placed in a group on a container control, such as a Panel or a GroupBox, and one of them is selected

Properties

- Appearance
- AutoCheck
- Checked
- Image

Method

- PerformClick
- Select
- Show

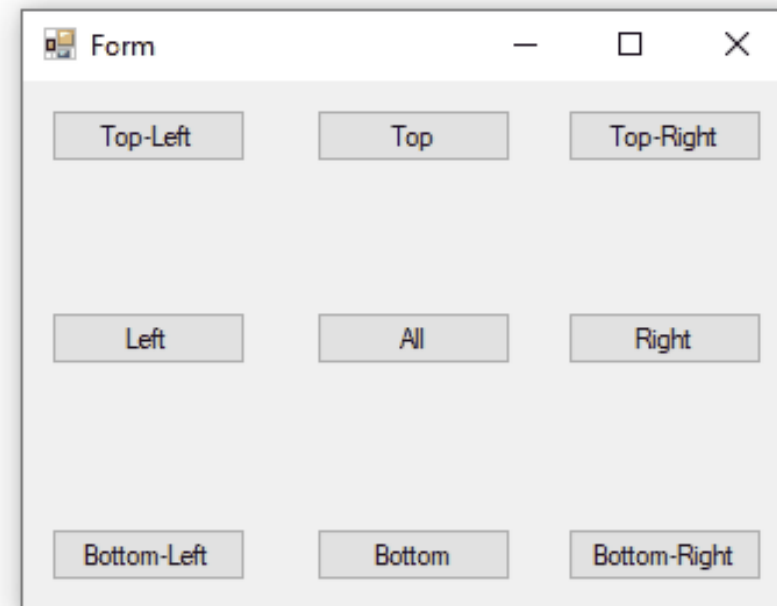
Event

- CheckedChanged
- Click

The screenshot shows a web form for configuring a car. It features two groups of radio buttons, each enclosed in a red rectangular box. The first group, on an orange background, contains three options: '2 Door' (selected), '4 Door', and 'Hatchback'. The second group, on a green background, contains three options: '4 Cylinder', '6 Cylinder', and '8 Cylinder' (selected). Below these groups is a section titled 'Options' containing three checked checkboxes: 'Air conditioning', 'Cruise Control', and 'Power Steering'. A 'Submit' button is located at the bottom right of the form.

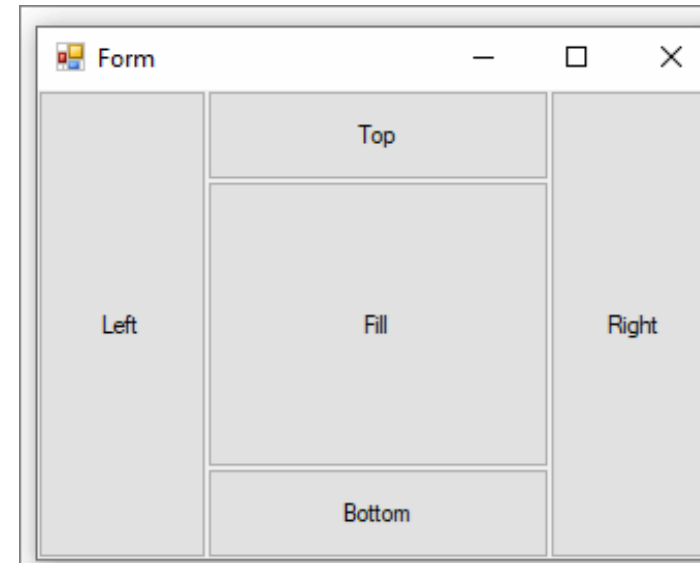
Position and Layout of Controls

- ◆ **Anchor:** Anchoring a control allows us to tie the control to one or more sides of the parent container. As the container changes in size, any child control will maintain its distance to the anchored side



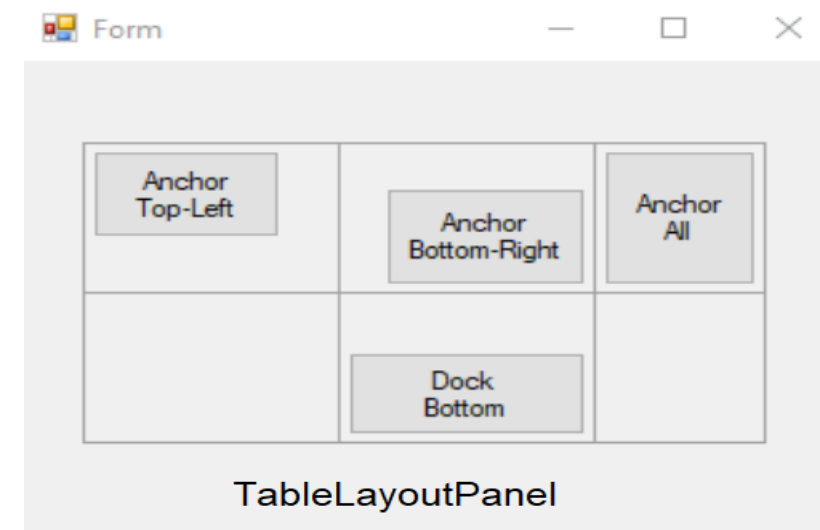
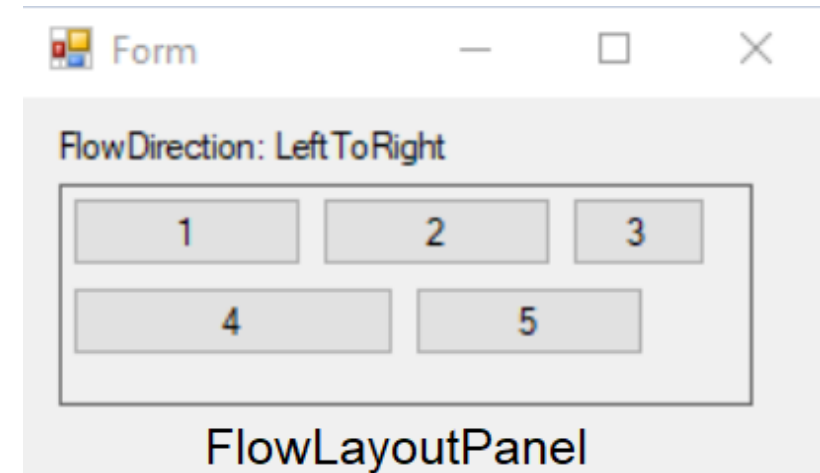
Position and Layout of Controls

- ◆ **Dock:** The Dock property sets which border of the control is aligned to the corresponding side of the parent, and how the control is resized within the parent
- ◆ When a control is docked, the container determines the space it should occupy and resizes and places the control. The width and height of the control are still respected based on the docking style



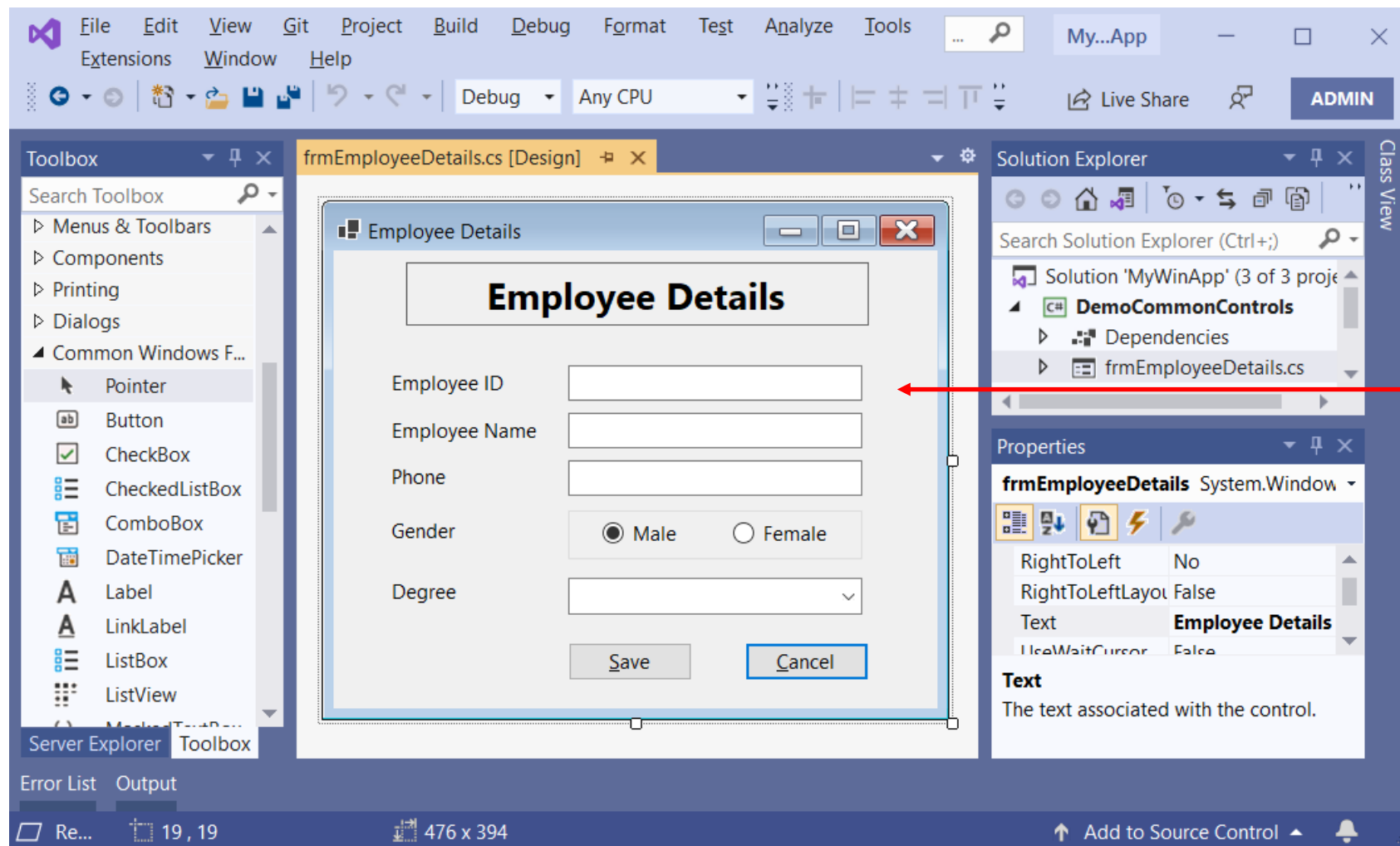
Position and Layout of Controls

- ◆ **Flow Layout:** The FlowLayoutPanel control arranges its contents in a horizontal or vertical flow direction. We can wrap the control's contents from one row to the next, or from one column to the next
- ◆ **Table layout:** The TableLayoutPanel control arranges its contents in a grid. Because the layout is done both at design time and run time, it can change dynamically as the application environment changes



Working with Controls Demonstration

1. Create a WinForm app named **DemoCommonControls** which includes a form named **frmEmployeeDetails** as follows:



2.Design UI as the following description table:

Object Type	Object name	Properties	Event Handlers
Label	lbTitle	BorderStyle: FixedSingle ; Font: 16.2pt, style=Bold	
Label	lbEmpID	Text : Employee ID	
Label	lbEmpName	Text : Employee Name	
Label	lbPhone	Text: Phone	
Label	lbGender	Text : Gender	
Label	lbDegree	Text: Degree	
TextBox	txtEmployeeID		
TextBox	txtEmployeeName		
MaskedTextBox	mtxtPhone	Mask: 000-0000000	
GroupBox	gbGender		

Object Type	Object name	Properties	Event Handlers
RadioButton	rdMale	Text: Male Checked: true	
RadioButton	rdFemale	Text: Female	
ComboBox	cboDegree	Text: ---Select Degree--- Items: Ph. D. Master Engineer Bachelor Technician	
Button	btnSave	Text: &Save	Click
Button	btnCancel	Text: &Cancel	Click
Form	frmEmployeeDetails	Text: Employee Details; StartPosition: CenterScreen AcceptButton: btnSave CancelButton: btnCancel	

3. Write codes for `frmEmployeeDetails.cs` then run project:

```
private void btnSave_Click(object sender, EventArgs e){
    string EmployeeID = txtEmployeeID.Text;
    string EmployeeName = txtEmployeeName.Text;
    string Phone = txtPhone.Text;
    string Gender = (rdFemale.Checked ? "Female" : "Male");
    string Degree = cboDegree.Text;
    StringBuilder builder = new StringBuilder();
    builder.Append($"EmployeeID:{EmployeeID}\n");
    builder.Append($"EmployeeName:{EmployeeName}\n");
    builder.Append($"Phone:{Phone}\n");
    builder.Append($"Gender:{Gender}\n");
    builder.Append($"Degree:{Degree}");
    MessageBox.Show(builder.ToString(), "Employee Details");
}

private void btnCancel_Click(object sender, EventArgs e)
{
    this.Close();
}
```

The screenshot shows two windows from a Windows application. The main window, titled 'Employee Details', contains a form with the following fields: 'Employee ID' (text box with 'EMP000001'), 'Employee Name' (text box with 'John'), 'Phone' (text box with '090-9090909'), 'Gender' (radio buttons for 'Male' and 'Female', with 'Male' selected), and 'Degree' (dropdown menu with 'Engineer' selected). At the bottom of the form are 'Save' and 'Cancel' buttons, both highlighted with red rectangles. To the right is a smaller summary window, also titled 'Employee Details', which displays the entered data: 'EmployeeID:EMP000001', 'EmployeeName:John', 'Phone:090-9090909', 'Gender:Male', and 'Degree:Engineer'. It has an 'OK' button at the bottom.

CheckBox

- ◆ A CheckBox control allows users to select single or multiple items from a list of items

Properties

- Checked
- CheckState
- ThreeState

Method

- Select
- Show

Event

- CheckStateChanged
- CheckStateChange
- Click

The screenshot shows a web form with two columns of radio button options and a section of checkbox options. The first column (orange background) has three options: '2 Door' (selected), '4 Door', and 'Hatchback'. The second column (green background) has three options: '4 Cylinder', '6 Cylinder', and '8 Cylinder' (selected). Below these is a section titled 'Options' containing three checked checkboxes: 'Air conditioning', 'Cruise Control', and 'Power Steering'. A 'Submit' button is located at the bottom right. A red rectangle highlights the 'Options' section.

CheckedListBox

- ◆ Displays a ListBox in which a check box is displayed to the left of each item

Properties

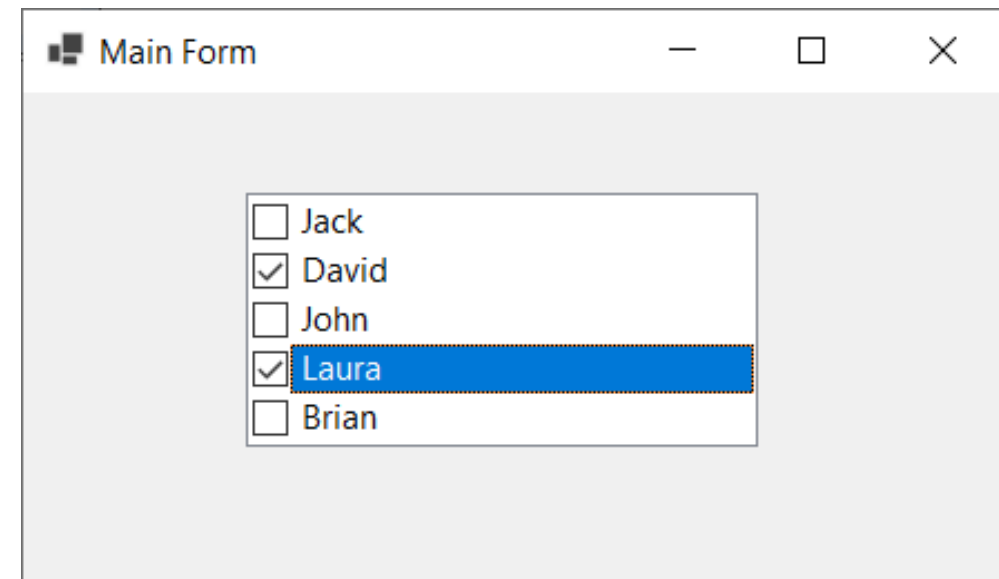
- CheckedIndices
- CheckedItems
- CheckOnClick
- Items
- SelectedValue
- SelectedItems
- SelectedItem

Method

- ClearSelected
- GetItemChecked
- GetItemCheckState
- GetItemText
- SetItemChecked
- SetItemCheckState

Event

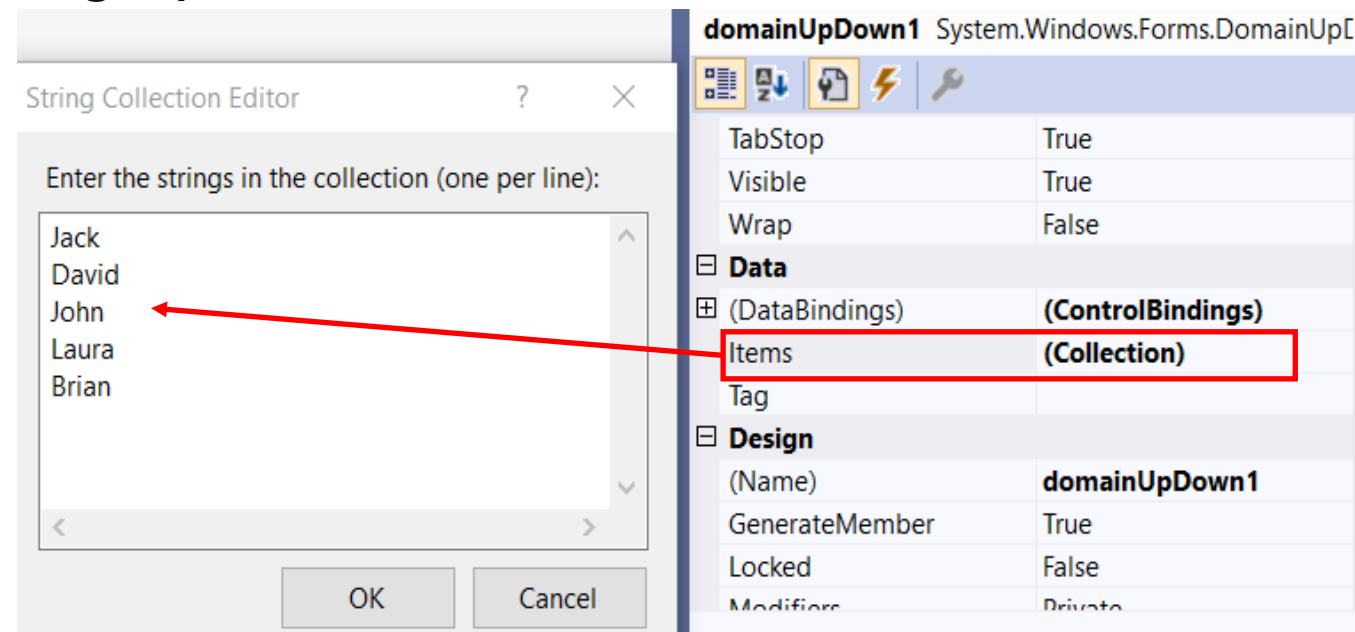
- ItemCheck
- MouseClick
- SelectedIndexChanged
- SelectedValueChanged



DomainUpDown

- A DomainUpDown control allows users to provide a spin (up/down) interface to move through pre-defined strings using up and down arrows

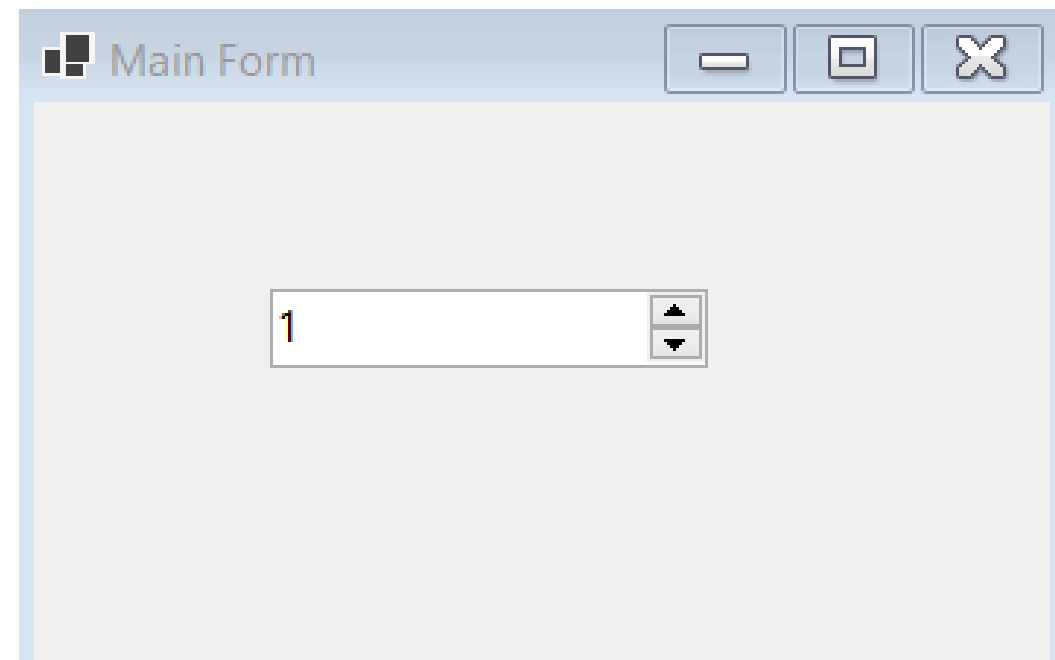
Properties	Items, Wrap, ReadOnly SelectedItem, SelectedIndex
Methods	DownButton, UpButton
Events	SelectedItemChanged



NumericUpDown

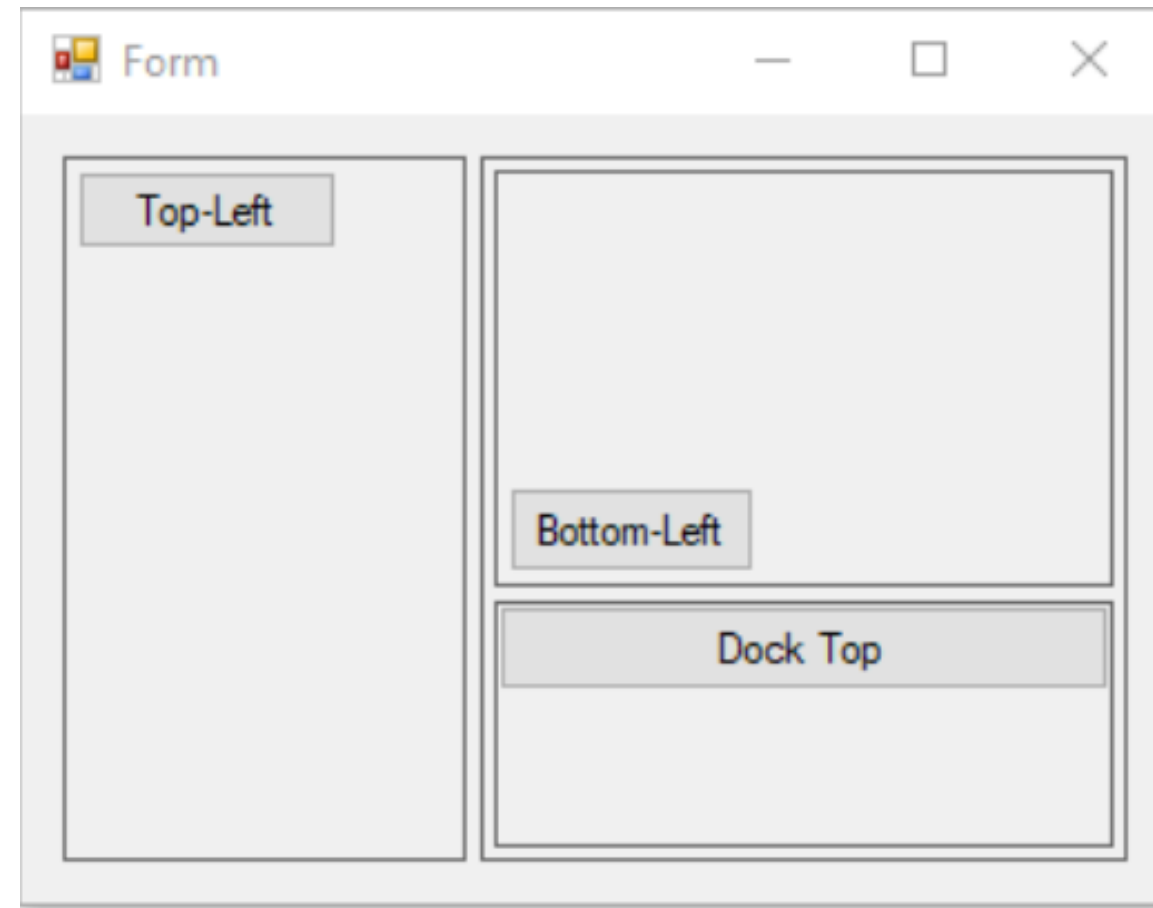
- ◆ A NumericUpDown control allows users to provide a spin (up/down) interface to move through pre-defined numbers using up and down arrows

Properties	Increment, Maximum, Minimum, Value DecimalPlaces ThousandsSeperator
Methods	DownButton UpButton
Events	ValueChanged



SplitContainer

- ◆ The SplitContainer control can be thought of as a composite control. It's two panels separated by a movable bar. When the mouse pointer is over the bar, the pointer changes shape to show that the bar is movable

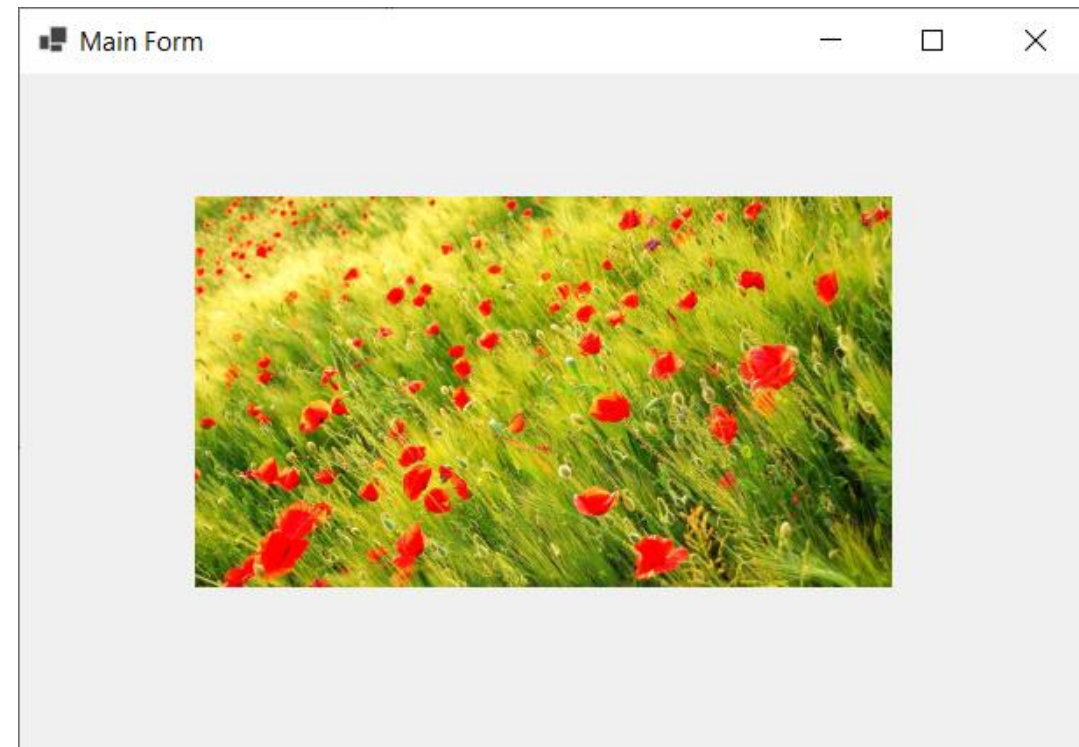


PictureBox

- ◆ PictureBox control is used to display images in Windows Forms

Properties	Image, ErrorImage, InitialImage, SizeMode
Methods	Load, LoadAsync
Events	Click, Leave, LoadCompleted

```
public partial class frmMain : Form {
    public frmMain()...
    private void frmMain_Load(object sender, EventArgs e){
        DisplayImage();
    }
    private void DisplayImage(){
        PictureBox imageControl = new PictureBox();
        imageControl.Width = 398;
        imageControl.Height = 223;
        imageControl.Location = new Point(100, 70);
        Bitmap image = new Bitmap("Images/MyImage1.jpg");
        imageControl.SizeMode = PictureBoxSizeMode.StretchImage;
        imageControl.Image = (Image)image;
        Controls.Add(imageControl);
    }
}
} //end frmMain class
```

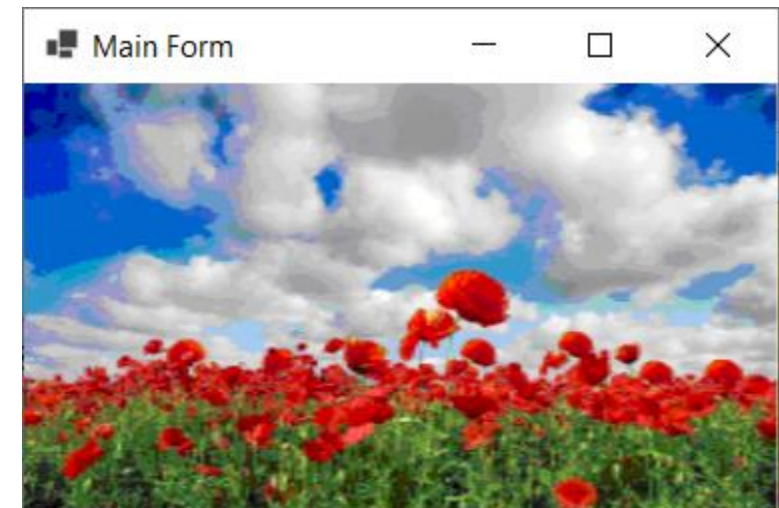
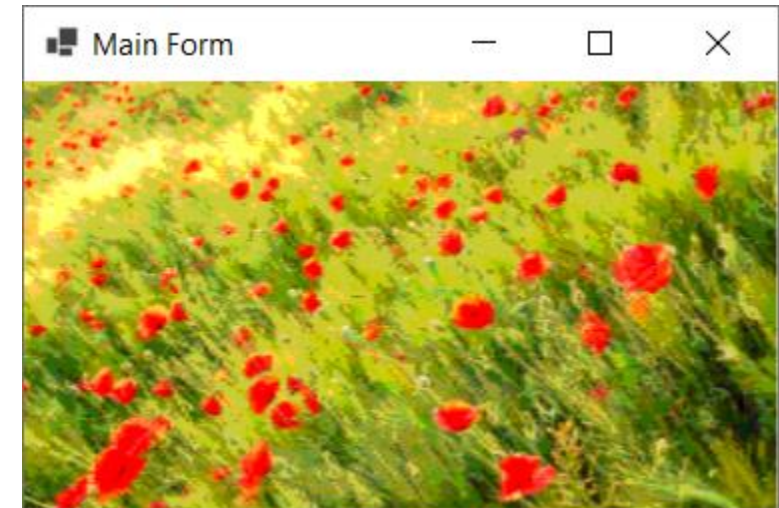


ImageList

- Provides methods to manage a collection of Image objects

Properties	Images, ColorDepth, ImageSize, Name
Methods	Draw
Events	RecreateHandle

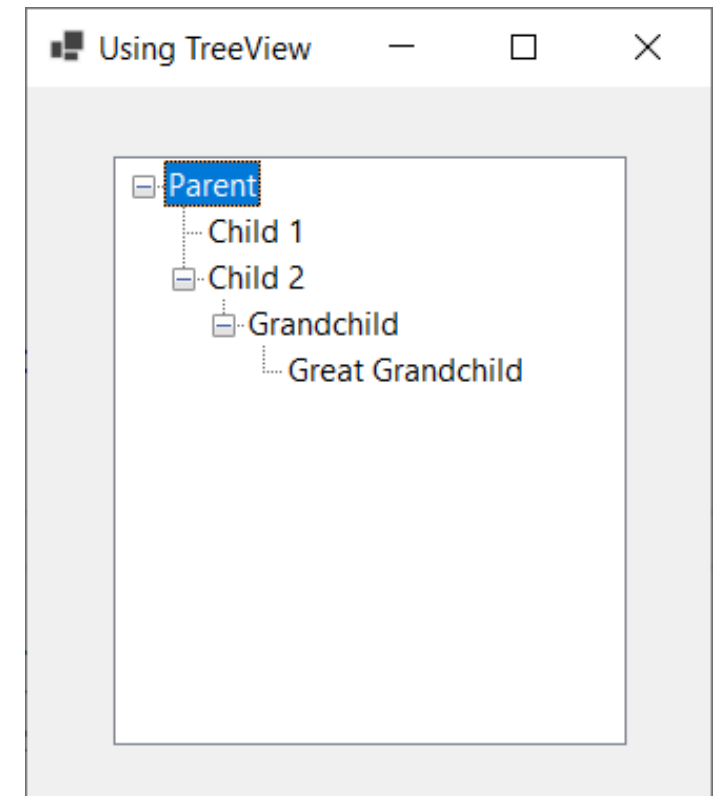
```
ImageList list = new ImageList();
Image image1 = Image.FromFile("Images/MyImage1.jpg");
Image image2 = Image.FromFile("Images/MyImage2.jpg");
list.Images.AddRange(new Image[] { image1, image2 });
list.ImageSize = new Size(200, 200);
```



TreeView

- ◆ Used for displaying data in a hierarchical maner
- ◆ Has three types of nodes : Root, Parent and Leaf

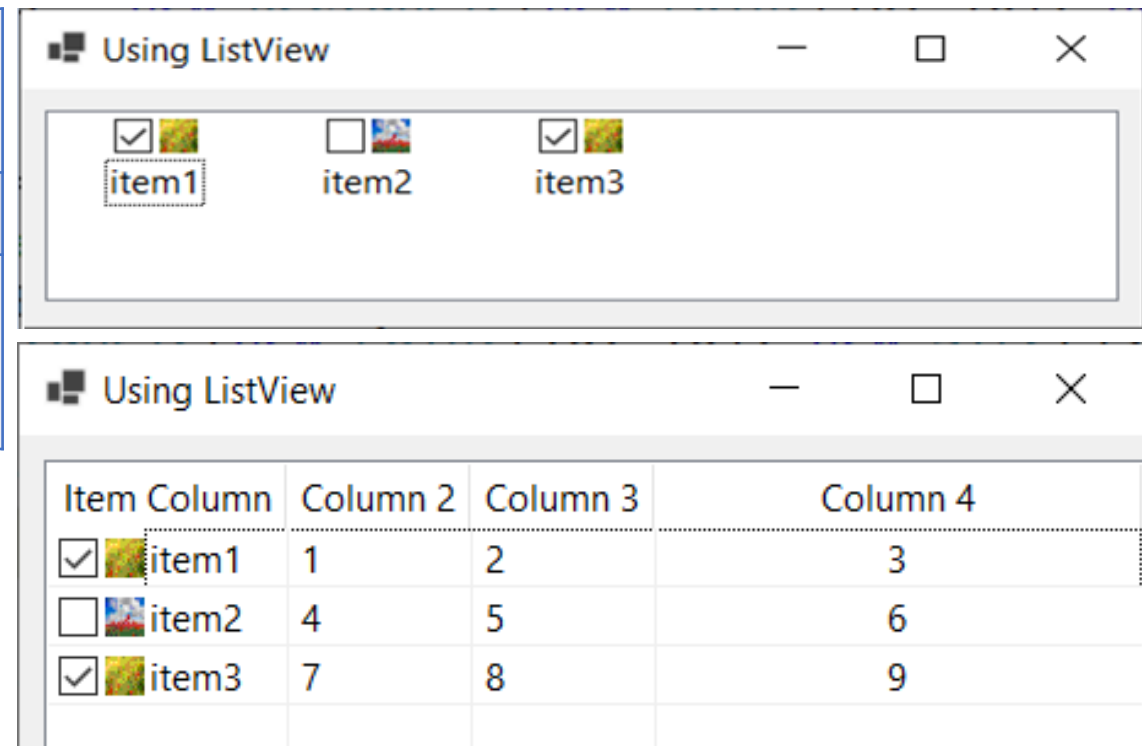
Properties	Nodes, TopNode, SelectedNode, ShowPlusMinus, ShowRootLines
Methods	CollapseAll, ExpandAll, GetNodeAt, GetNodeCount
Events	NodeMouseClicked, ItemDrag, AfterCollapse, AfterExpand, AfterSelect



List View

- Used to display a collection of items in a list. The ListView control is an ItemsControl that is derived from ListBox
- Types of ListView: Tile, List, Details, SmallIcon, LargeIcon

Properties	Columns, Items, View, MultiSelect, Sort, SelectedItems, SelectedIndices
Methods	GetItemAt, Clear, ArrangeIcons
Events	ColumnClick, SelectedIndexChanged, ItemCheck, ItemSelectionChanged



Listview Demo

```
public partial class frmListView : Form{
    public frmListView()...

    private void CreateMyListView() {
        // Create a new ListView control.
        ListView listView1 = new ListView();
        listView1.Bounds = new Rectangle
            (new Point(10, 10), new Size(450, 200));
        // Set the view to show details.
        listView1.View = View.Details;
        // Allow the user to edit item text.
        listView1.LabelEdit = true;
        // Allow the user to rearrange columns.
        listView1.AllowColumnReorder = true;
        // Display check boxes.
        listView1.CheckBoxes = true;
        // Select the item and subitems when selection is made.
        listView1.FullRowSelect = true;
        // Display grid lines.
        listView1.GridLines = true;
        // Sort the items in the list in ascending order.
        listView1.Sorting = SortOrder.Ascending;

        // Create three items and three sets of subitems for each item.
        ListViewItem item1 = new ListViewItem("item1", 0);
        // Place a check mark next to the item.
        item1.Checked = true;
        item1.SubItems.Add("1");
        item1.SubItems.Add("2");
        item1.SubItems.Add("3");
        ListViewItem item2 = new ListViewItem("item2", 1);
        item2.SubItems.Add("4");
        item2.SubItems.Add("5");
        item2.SubItems.Add("6");
        ListViewItem item3 = new ListViewItem("item3", 0);
        // Place a check mark next to the item.
        item3.Checked = true;
        item3.SubItems.Add("7");
        item3.SubItems.Add("8");
        item3.SubItems.Add("9");
    }
}
```

Listview Demo

```
// Create columns for the items and subitems.
// Width of -2 indicates auto-size.
listView1.Columns.Add("Item Column", -2, HorizontalAlignment.Left);
listView1.Columns.Add("Column 2", -2, HorizontalAlignment.Left);
listView1.Columns.Add("Column 3", -2, HorizontalAlignment.Left);
listView1.Columns.Add("Column 4", -2, HorizontalAlignment.Center);

//Add the items to the ListView.
listView1.Items.AddRange(new ListViewItem[] { item1, item2, item3 });

// Create two ImageList objects.
ImageList imageListSmall = new ImageList();
ImageList imageListLarge = new ImageList();




// Initialize the ImageList objects with bitmaps.
imageListSmall.Images.Add(Bitmap.FromFile("Images/MyImage1.jpg"));
imageListSmall.Images.Add(Bitmap.FromFile("Images/MyImage2.jpg"));
imageListLarge.Images.Add(Bitmap.FromFile("Images/MyImage1.jpg"));
imageListLarge.Images.Add(Bitmap.FromFile("Images/MyImage2.jpg"));
```

```
//Assign the ImageList objects to the ListView.
listView1.LargeImageList = imageListLarge;
listView1.SmallImageList = imageListSmall;

// Add the ListView to the control collection.
this.Controls.Add(listView1);
} //end method

//-----
private void frmListView_Load(object sender, EventArgs e)
{
    CreateMyListView();
}
```

Using ListView

Item Column	Column 2	Column 3	Column 4
<input checked="" type="checkbox"/>  item1	1	2	3
<input type="checkbox"/>  item2	4	5	6
<input checked="" type="checkbox"/>  item3	7	8	9

DataGridView

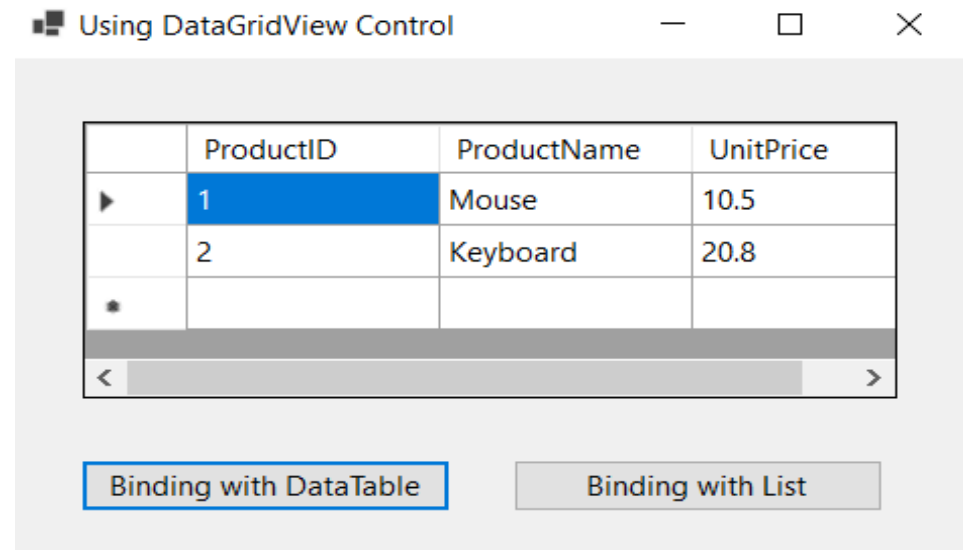
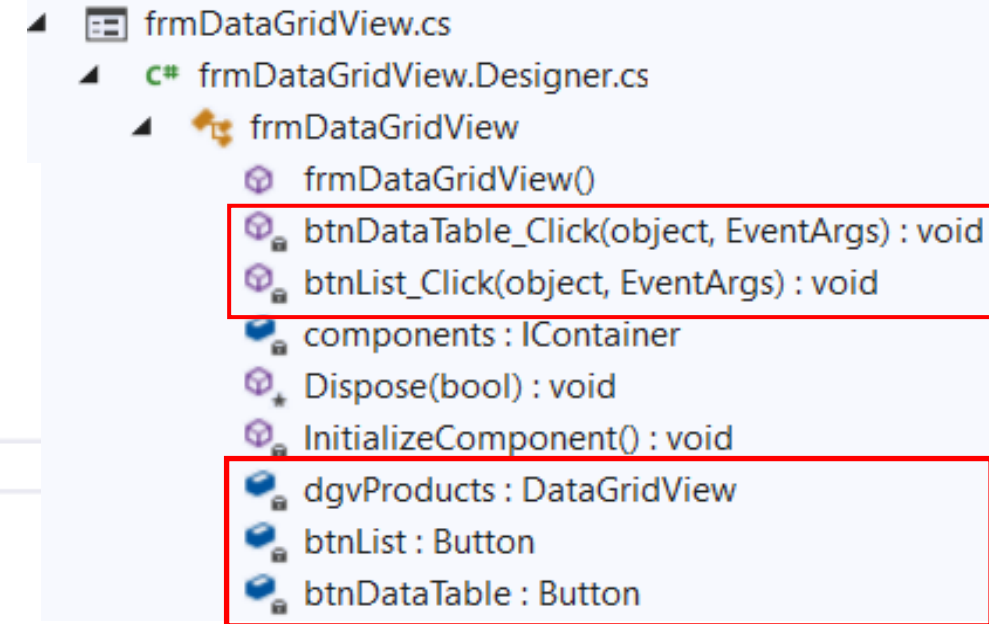
- ◆ The DataGridView control provides a powerful and flexible way to display data in a tabular format
- ◆ We can use the DataGridView control to show read-only views of a small amount of data, or we can scale it to show editable views of very large sets of data
- ◆ The DataGridView control supports display and edit tabular data from many different kinds of data sources. Binding data to the DataGridView control is straightforward and intuitive, and in many cases it is as simple as setting the DataSource property

DataGridView

- ◆ The DataGridView control supports the standard Windows Forms data binding model, so it will bind to instances of classes described in the following list:
 - Any class that implements the IList interface, including one-dimensional arrays
 - Any class that implements the IListSource interface, such as the DataTable and DataSet classes
 - Any class that implements the IBindingList interface, such as the BindingList<> class
 - Any class that implements the IBindingListView interface, such as the BindingSource class

DataGridView

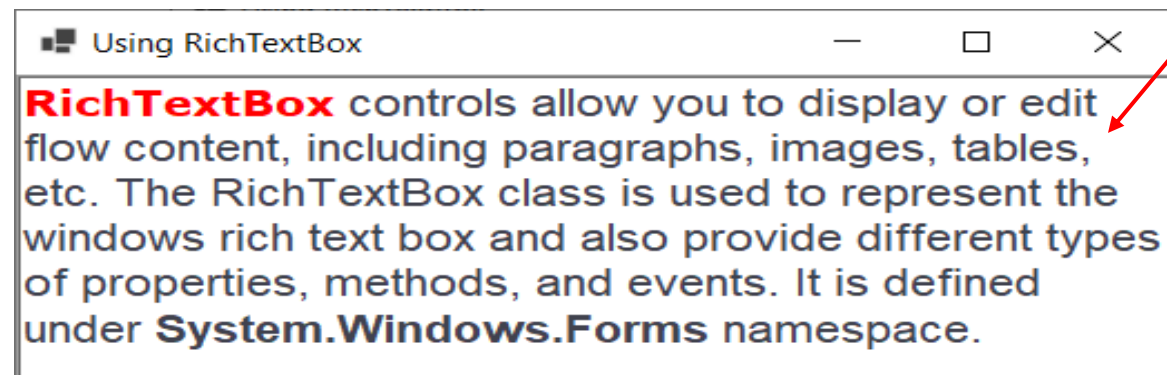
```
public partial class frmDataGridView : Form{
    public frmDataGridView()...
    private void btnDataTable_Click(object sender, EventArgs e) {
        DataTable products = new DataTable();
        products.Columns.Add("ProductID",typeof(int));
        products.Columns.Add("ProductName", typeof(string));
        products.Columns.Add("UnitPrice", typeof(decimal));
        products.Rows.Add(1, "Mouse",10.5);
        products.Rows.Add(2, "Keyboard",20.8);
        dgvProducts.DataSource = products;
    }//end btnDataTable_Click
    private void btnList_Click(object sender, EventArgs e){
        List<dynamic> products = new List<dynamic>{
            new {ProductID=3, ProductName="Speaker",UnitPrice=14.9},
            new {ProductID=4, ProductName="Monitor",UnitPrice=50.5}
        };
        dgvProducts.DataSource = products;
    }//end btnList_Click
}
```



RichTextbox

- RichTextBox control is a textbox that gives us rich text editing controls and advanced formatting features also includes loading rich text format (RTF) files

Properties	Font, ScrollBars, SelectedText, SelectionFont, SelectionLength, Text, WordWrap
Methods	AppendText, Copy, Paste, Redo, Undo
Events	Click, HScroll, Vscroll, SelectionChanged



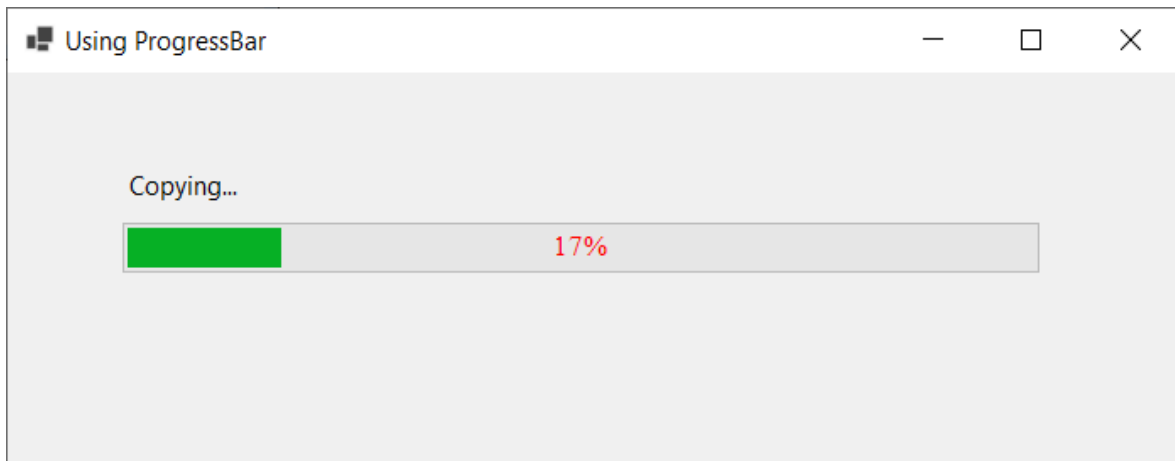
```
private void frmRichTextBox_Load(object sender, EventArgs e){
    CreateMyRichTextBox();
}

public void CreateMyRichTextBox(){
    RichTextBox rtbData = new RichTextBox();
    rtbData.Dock = DockStyle.Fill;
    rtbData.LoadFile("MyData.rtf");
    rtbData.Find("RichTextBox", RichTextBoxFinds.MatchCase);
    rtbData.SelectionFont = new Font("Verdana", 12, FontStyle.Bold);
    rtbData.SelectionColor = Color.Red;
    rtbData.SaveFile("MyData.rtf", RichTextBoxStreamType.RichText);
    this.Controls.Add(rtbData);
}
```

ProgressBar

- A ProgressBar control is used to represent the progress of a lengthy operation that takes time where a user must wait for the operation to be finished

Properties	Maximum; Minimum; Step; Style; Value
Methods	Increment, PerformStep



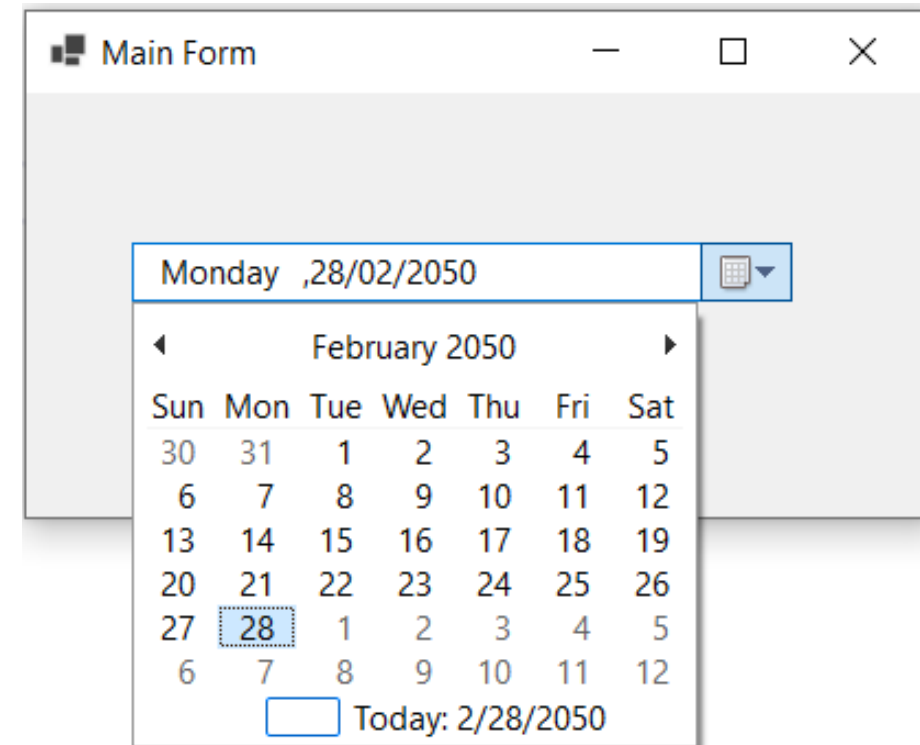
```
//Copy files method
private void CopyWithProgress(string[] filenames){
    // Set Minimum to 1 to represent the first file being copied.
    pBMain.Minimum = 1;
    // Set Maximum to the total number of files to copy.
    pBMain.Maximum = filenames.Length;
    // Set the initial value of the ProgressBar.
    pBMain.Value = 1;
    // Set the Step property to a value of 1 to represent each file being copied.
    pBMain.Step = 1;
    // Loop through all files to copy.
    for (int x = 1; x <= filenames.Length; x++){
        // Copy the file and increment the ProgressBar if successful.
        if (CopyFile(filenames[x - 1]) == true){
            // Perform the increment on the ProgressBar.
            pBMain.PerformStep();
        }
    }
}
```

DateTimePicker

- The Windows Forms DateTimePicker control allows the user to select a single item from a list of dates or times

Properties	Format, MinDate, MaxDate, ShowCheckBox, Value, CustomFormat
Methods	Focus, Show
Events	Click, FormatChanged, ValueChanged

```
private void frmMain_Load(object sender, EventArgs e)
{
    dateTimePicker1.Format = DateTimePickerFormat.Custom;
    dateTimePicker1.CustomFormat = "dddd, dd/MM/yyyy";
}
```

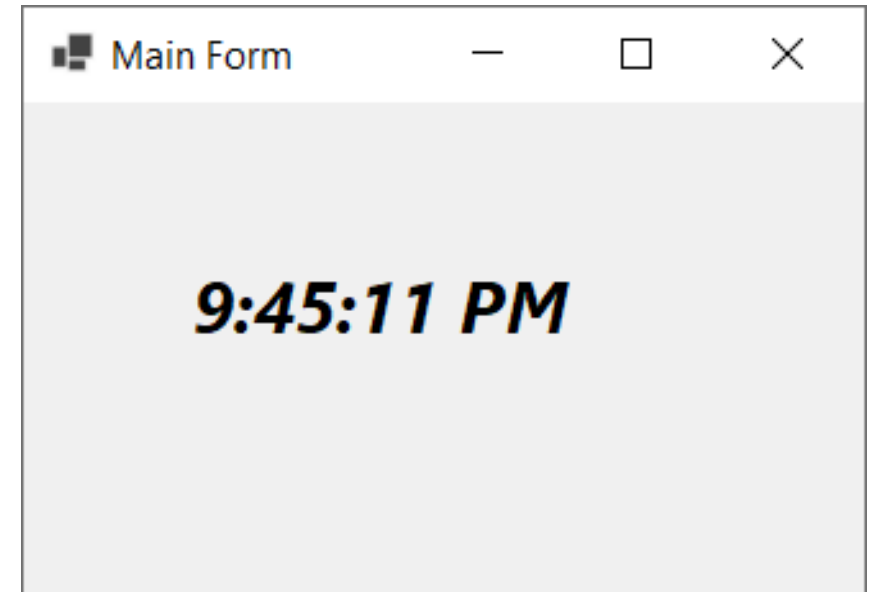


Timer

- ◆ The Timer class in C# represents a Timer control that executes a code block at a specified interval of time repeatedly

Properties	Enable, Interval
Methods	Start, Stop
Events	Tick

```
private void frmMain_Load(object sender, EventArgs e){
    Timer timer = new Timer();
    timer.Interval = 1000;
    timer.Tick += Timer_Tick;
    timer.Start();
}
private void Timer_Tick(object sender, EventArgs e) {
    lbCurrentTime.Text = DateTime.Now.ToLongTimeString();
}
```



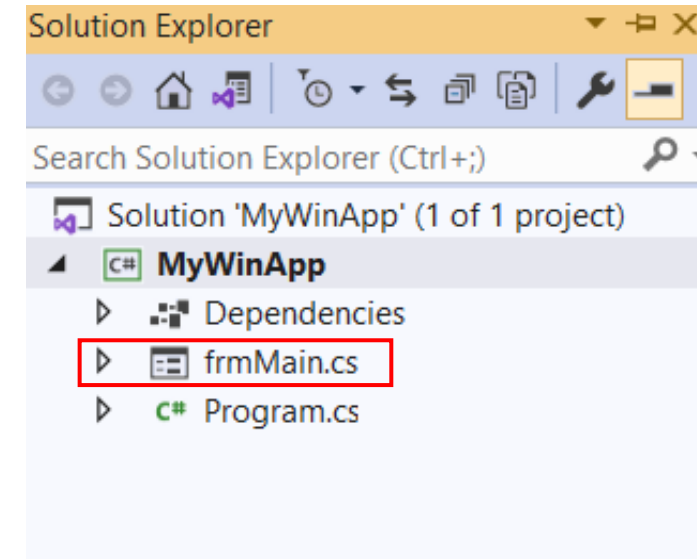
MenuStrips

- ◆ Common UI elements that may be contained within a MenuStrip:
 - ToolStripMenuItem: A traditional menu item
 - ToolStripComboBox: An embedded ComboBox
 - ToolStripSeparator: A simple line that separates content
 - ToolStripTextBox: An embedded TextBox
- ◆ When the ampersand character (&) is placed before a letter in a menu item, it denotes the item's shortcut key. In this example, we are creating &File and E&xit; therefore, the user may activate the Exit menu by pressing Alt+F, and then X

MenuStrips Demonstration

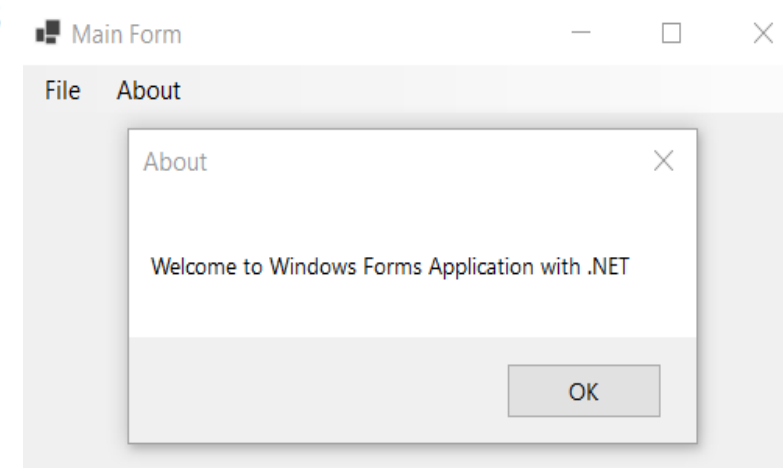
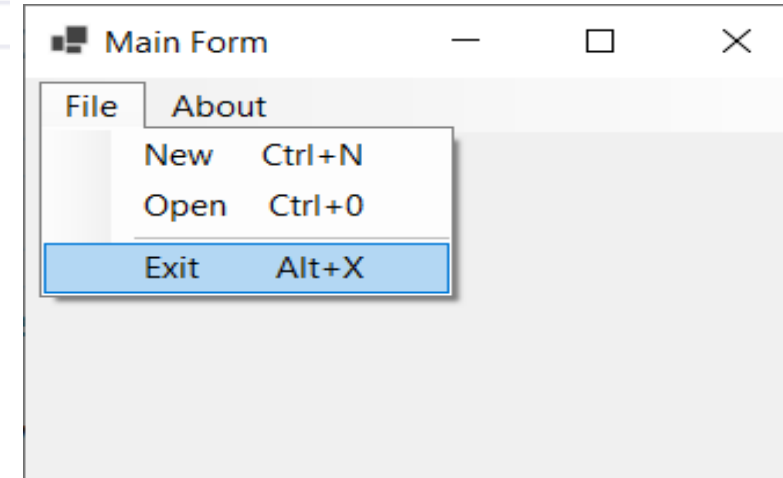
- ◆ Create a WinForm app named **MyApp** which includes a form named **frmMain** then writes codes as follows:

```
public partial class frmMain : Form {
    public frmMain() {
        ...
    }
    private void frmMain_Load(object sender, EventArgs e) {
        CreateMainMenu();
    }
    public void CreateMainMenu() {
        MenuStrip mainMenu = new MenuStrip();
        this.Controls.Add(mainMenu);
        this.MainMenuStrip = mainMenu;
        ToolStripMenuItem menuFile = new ToolStripMenuItem("&File");
        ToolStripMenuItem mnuNew = new ToolStripMenuItem("&New");
        ToolStripMenuItem mnuOpen = new ToolStripMenuItem("&Open");
        ToolStripSeparator separator = new ToolStripSeparator();
        ToolStripMenuItem mnuExit = new ToolStripMenuItem("&Exit");
        ToolStripMenuItem mnuAbout = new ToolStripMenuItem("&About");
        ToolStripMenuItem mnuViewAbout = new ToolStripMenuItem("&View About");
        // mainMenu
        mainMenu.Items.AddRange(new ToolStripItem[] { menuFile, mnuAbout });
        // menuFile
        menuFile.DropDownItems.AddRange(new ToolStripItem[] { mnuNew, mnuOpen,
            separator, mnuExit });
        // mnuAbout
        mnuAbout.DropDownItems.AddRange(new ToolStripItem[] { mnuViewAbout });
        // mnuNew
        mnuNew.ShortcutKeys = (Keys)((Keys.Control | Keys.N));
    }
}
```



MenuStrips Demonstration

```
// mnuOpen
mnuOpen.ShortcutKeys = (Keys)((Keys.Control | Keys.O));
// mnuExit
mnuExit.ShortcutKeys = (Keys)((Keys.Alt | Keys.X));
mnuExit.Click += new EventHandler(mnuExit_Click);
// mnuViewAbout
mnuViewAbout.ShortcutKeys = Keys.F1;
mnuViewAbout.Click += new EventHandler(mnuViewDetail_Click);
} //end CreateMainMenu
//Click EventHandler
private void mnuViewDetail_Click(object sender, EventArgs e) {
    MessageBox.Show("Welcome to Windows Forms Application with .NET", "About");
}
private void mnuExit_Click(object sender, EventArgs e) {
    Application.Exit();
}
} //end frmMain class
```

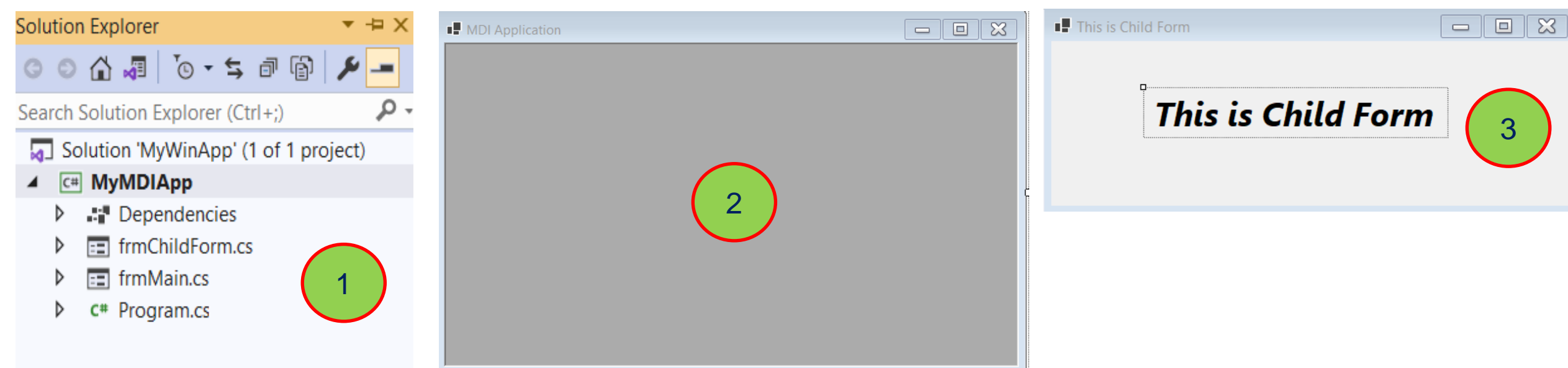


Multiple-Document Interface(MDI) Application

- ◆ Multiple-document interface (MDI) applications enable you to display multiple documents at the same time, with each document displayed in its own window
- ◆ The foundation of a Multiple-Document Interface (MDI) application is the MDI parent form. This is the form that contains the MDI child windows, which are the sub-windows wherein the user interacts with the MDI application
- ◆ MDI applications follow a parent form and child form relationship model. MDI applications allow us to open, organize, and work with multiple documents at the same time by opening them under the context of the MDI parent form
- ◆ MDI applications often have a Window menu item with submenus for switching between windows or documents

MDI Application Demonstration

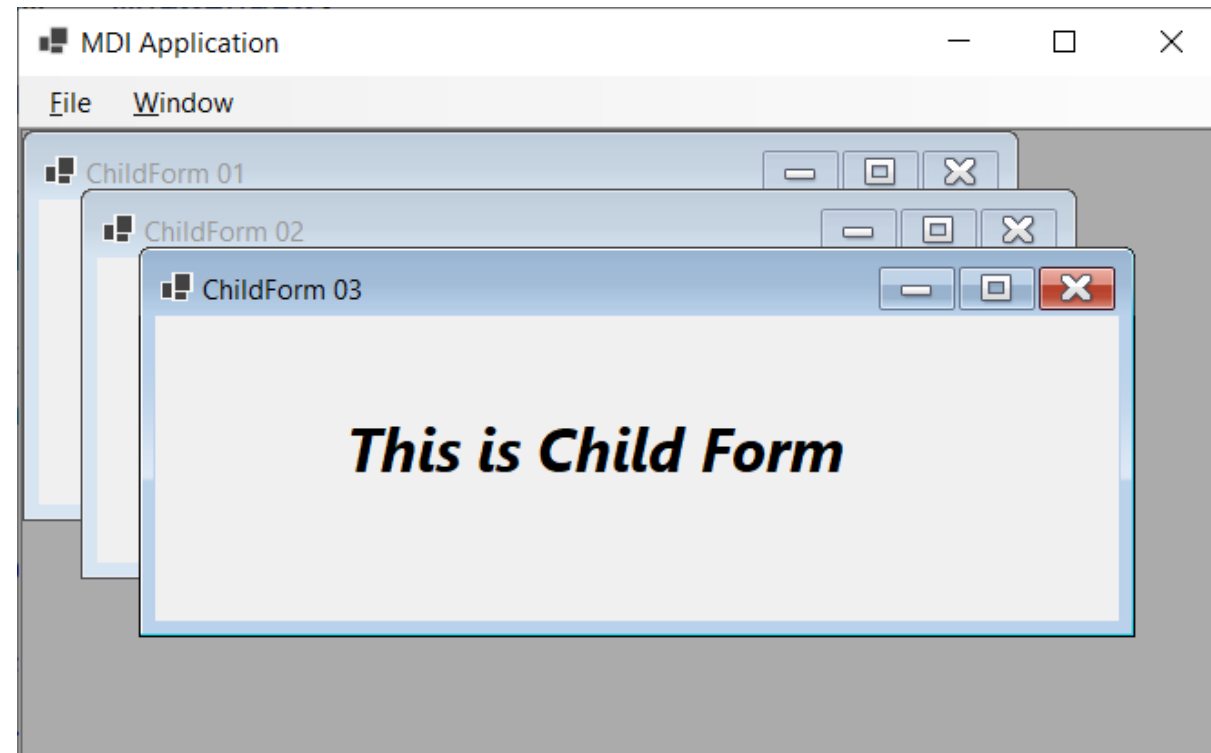
1. Create a WinForm app named **MyMDIApp** which includes a form named **frmMain**
2. On the **frmMain**, set the "**IsMdiContainer**" property to **True** (the default value is False).
Notice that the background color of the form has changed to dark gray
3. On the **MyMDIApp** project, right-click, and select **Add | Form(Windows form)** named **frmChildForm** and click **Add** then design user interface as follows :



4. Write codes for **frmMain.cs** then run project:

```
public partial class frmMain : Form{
    public frmMain()...
    int counter = 1;
    private void frmMain_Load(object sender, EventArgs e){
        CreateMyMainMenu();
    }
    public void CreateMyMainMenu(){
        MenuStrip mainMenu = new MenuStrip();
        this.Controls.Add(mainMenu);
        this.MainMenuStrip = mainMenu;
        ToolStripMenuItem menuFile = new ToolStripMenuItem("&File");
        ToolStripMenuItem mnuOpen = new ToolStripMenuItem("&Open");
        ToolStripSeparator separator = new ToolStripSeparator();
        ToolStripMenuItem mnuExit = new ToolStripMenuItem("&Exit");
        ToolStripMenuItem mnuWindow = new ToolStripMenuItem("&Window");
        // mainMenu
        mainMenu.Items.AddRange(new ToolStripItem[] { menuFile, mnuWindow });
        mainMenu.MdiWindowListItem = mnuWindow;
        // menuFile
        menuFile.DropDownItems.AddRange(new ToolStripItem[] {mnuOpen,separator,mnuExit});
        // mnuOpen
        mnuOpen.ShortcutKeys = (Keys)((Keys.Control | Keys.O));
        mnuOpen.Click += new EventHandler(mnuOpen_Click);
    }
}
```

```
// mnuExit
mnuExit.ShortcutKeys = (Keys)((Keys.Alt | Keys.X));
mnuExit.Click += new EventHandler(mnuExit_Click);
} //end CreateMainMenu
//Show frmChild form
private void mnuOpen_Click(object sender, EventArgs e) {
    frmChildForm childForm = new frmChildForm();
    childForm.Text = $"ChildForm {counter++:D2}";
    childForm.MdiParent = this;
    childForm.Show();
} //end mnuOpen_Click
private void mnuExit_Click(object sender, EventArgs e) {
    Application.Exit();
}
} //end MainForm
```



Lab and Assignment

1. Do Hands-on Lab:

[Lab_01_AutomobileManagement_Using_LINQ and WinForms.pdf](#)

2. Do Assignment:

[Assignment_01_MemberManagement.pdf](#)

Summary

- ◆ Concepts were introduced:
 - Overview Windows Forms (WinForm)
 - Explain about type Controls and Components in WinForm
 - Working with Common Controls: Label, TextBox, MaskedTextBox, etc
 - Working with Menu, ProgressBar and Timer in WinForm
 - Explain about Position and Layout in WinForm
 - Demo create Windows Forms Application
 - Demo creating and using Menu in WinForm
 - Demo create MDI Application in WinForm