
Winforms
Lesson 01:Working with
Windows Applications
Capgemini

References

- Microsoft.com/msdn
- MCTS Self Paced Training Kit (70-526)

Lesson Objectives

- In this lesson, you will learn:
  - The difference between Console and Windows Application
  - The challenges for Windows Applications
  - Event Handling Model in .NET
  - Different controls and their properties
  - The creation of MDI Applications and Menus in C#

Windows Applications

Introduction

- In the past, creating Windows applications was a challenging endeavor.
- The .NET library contains an entire subsystem that supports Windows Forms, which greatly simplifies the creation of a Windows program.
- The Windows applications are much easier to create by using C# and the System. Windows. Forms library.

Windows Applications:
C# and the .NET Framework's Forms library offer a fully object-oriented way to approach Windows programming.
Instead of providing just a wrapper around the API, the Forms library defines a streamlined, integrated, logically consistent way of managing the development of a Windows application. This level of integration is made possible by the unique features of the C# language, such as delegates and events. Furthermore, because of C#'s use of garbage collection, the troubling problem of "memory leaks" especially has been nearly eliminated.

C# Console Application

Explanation

- Library imports
- Class and namespace definitions

```
using System;
class HelloWorld
{
   static void Main(string[] args)

   {
      Console.WriteLine("Hello World!");
   }
}
```

C# Console Application:
C# Console Application has the following characteristics:

- No visual component
- Only text input and output
- Run under Command Prompt or DOS Prompt

Really Simple C# Windows Application

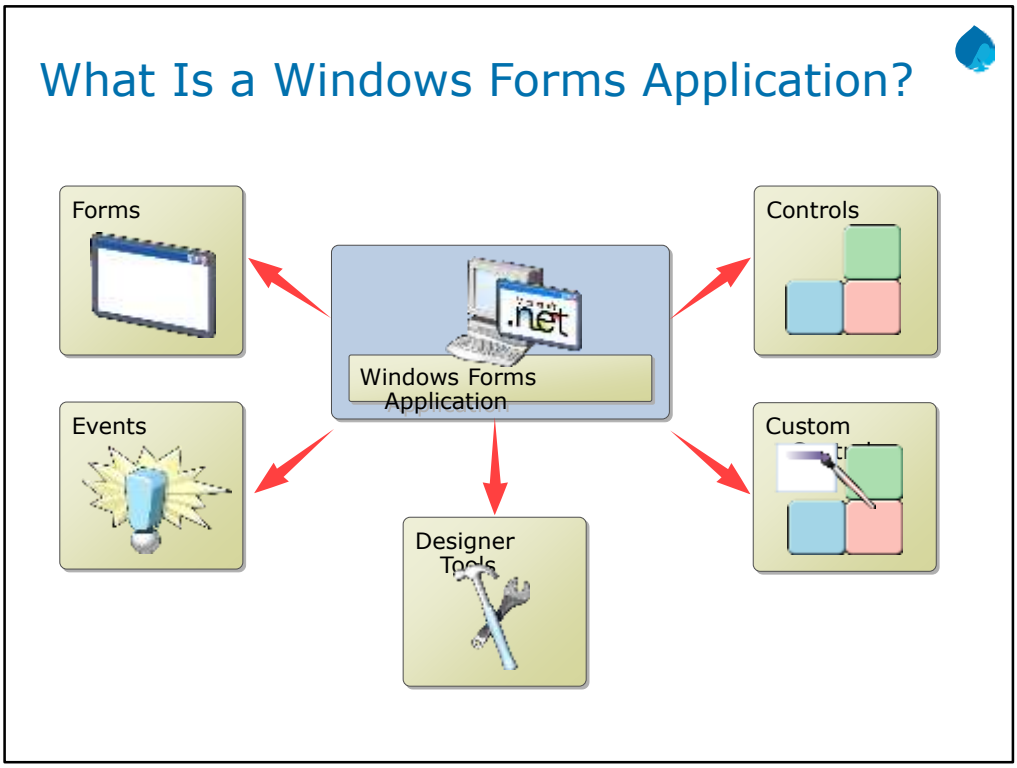
Explanation

- Library imports
- Class and namespace definitions

```
using System;
using System.Windows.Forms;
class HelloWorld
{
    static void Main(string[] args)
    {
        MessageBox.Show("Hello World!");
    }
}
```

C# Windows Application:
It forms with many different input and output types.
It contains Graphical User Interfaces (GUI).
GUIs make the input and output more user friendly!
Message boxes:

They are used within the System.Windows.Forms namespace.
They are used to prompt or display information to the user.


What Is a Windows Forms Application?
Forms
Controls
Windows Forms Application
Events
Custom
Designer Tools

Typical C# Window Application

Explanation

- Library imports
- Class and namespace definitions

```
using System;
using System.Windows.Forms;
class HelloWorld : System.Windows.Forms.Form
{
   static void Main(string[] args)
   {
      Application.Run(new HelloWorld());
   }
}
```

This code creates a Window Form of default size 300x300, with no title text.

It uses the default Hello World constructor inherited from the Form class

Typical C# Window Application:
A form is created by instantiating an object of the Form class, or of any class derived from Form.
A form contains significant functionality of its own, and it inherits additional functionality. Two of its most important base classes are System.ComponentModel.Component and System.Windows.Forms.Control. The Control class defines features common to all Windows controls. Since Form inherits Control, it too is a control. This fact allows forms to be used to create controls.

Explanation

- Library imports
  - using System;
  - using System.Drawing;
  - using System.Windows.Forms;
- Class and namespace definitions

New library that defines the Size class!

Explanation

```
class WinSkel : System.Windows.Forms.Form
{
   public WinSkel() {
     Size = new Size(300,100);
     Text = "HelloWorld";
   }
[STAThread]
   static void Main(string[] args)
   {
      Application.Run(new WinSkel());
   }
}
```

Typical C# Window Application (contd.):
Let us examine this program line by line.
First, notice that both System and System.Windows.Forms are included. System is needed because of the STAThread attribute that precedes Main( ). The System.Windows.Forms supports the Windows forms subsystem, as just explained. Next, a class called WinSkel is created. It inherits Form. Thus, WinSkel defines a specific type of form. In this case, it is a minimal form.

Inside the WinSkel constructor, there is the following line of code:

Text = "HelloWorld";

Text is the property that sets the title of the window. Thus, this assignment causes the title bar in the window to contain "HelloWorld". Text is defined as shown below:

public override string Text { get; set; } Text is inherited from Control.

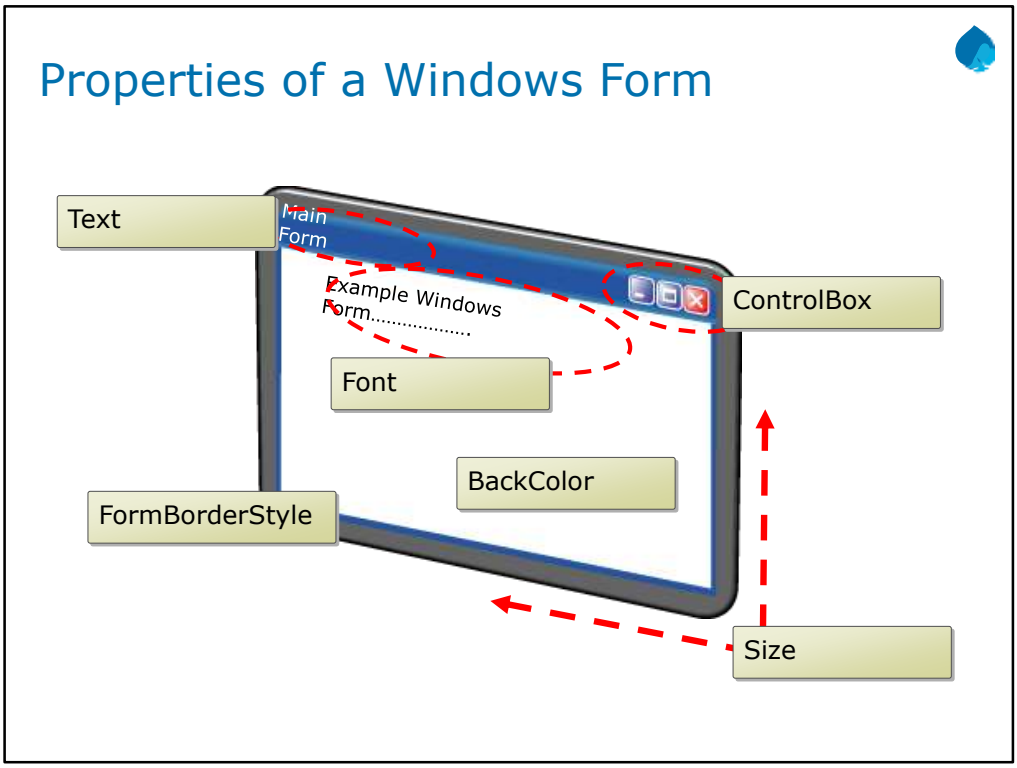
Next is the Main( ) method. It is the method at which program execution begins. Notice, however, that it is preceded by the STAThread property. As a general rule, the Main( ) method of a Windows program should have this property. It sets the threading model for the program to a single-threaded apartment (STA). Inside Main( ), a WinSkel object is created. This object is then passed to the Run( ) method defined by the Application class.

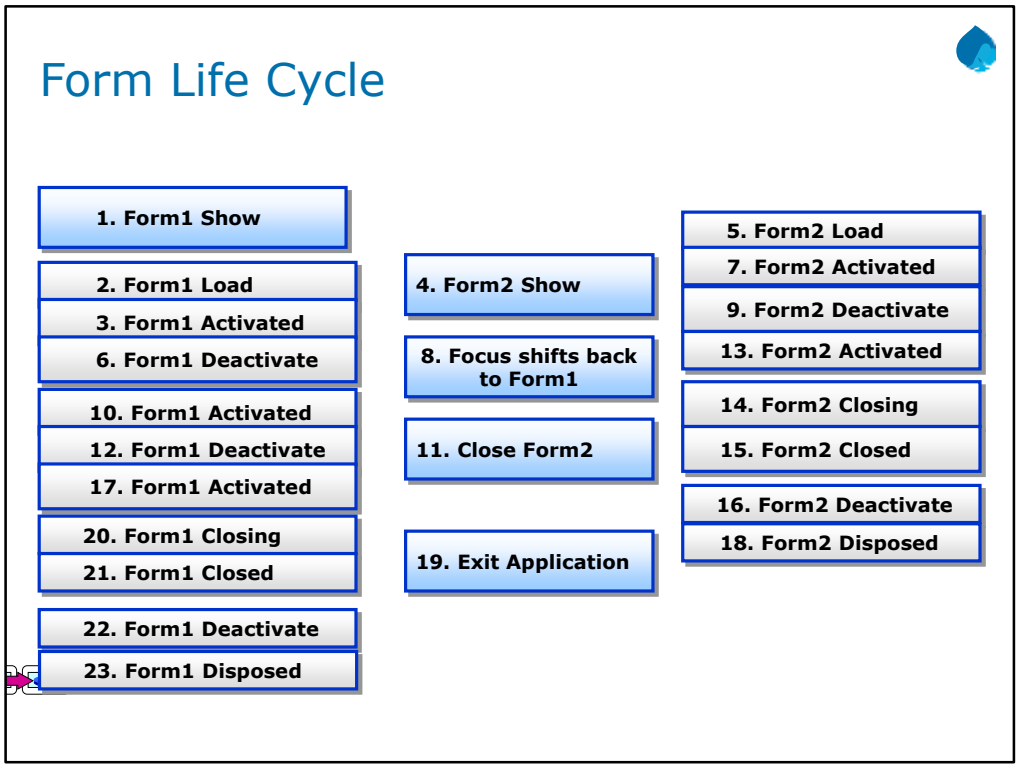
Application.Run(new Winskel());

This starts the window running. The Application class is defined within System.Windows.Forms, and it encapsulates aspects common to all Windows applications. The Run( ) method used by the skeleton is shown here:

public static void Run(Form ob)

It takes a reference to a form as a parameter. Since WinSkel inherits Form, an object of type WinSkel can be passed to Run( ).


Properties of a Windows Form
Text
Main Form
Example Windows Form.....
ControlBox
Font
BackColor
FormBorderStyle
Size


Form Life Cycle
1. Form1 Show
2. Form1 Load
3. Form1 Activated
6. Form1 Deactivate
10. Form1 Activated
12. Form1 Deactivate
17. Form1 Activated
20. Form1 Closing
21. Form1 Closed
22. Form1 Deactivate
23. Form1 Disposed
4. Form2 Show
8. Focus shifts back to Form1
11. Close Form2
19. Exit Application
5. Form2 Load
7. Form2 Activated
9. Form2 Deactivate
13. Form2 Activated
14. Form2 Closing
15. Form2 Closed
16. Form2 Deactivate
18. Form2 Disposed

Demo

- Demo on creating Windows Form based Application

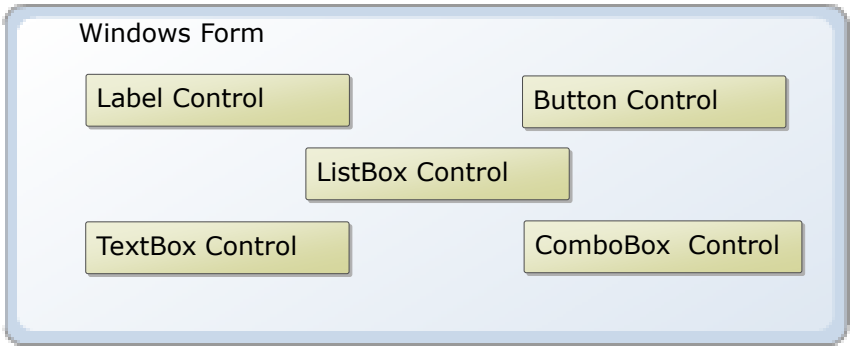
Challenges for Window Applications

Explanation

- What are basic inputs?
  - User action, system actions are basic inputs.
  - Some inputs come directly from user actions, some are indirect and implicit.
- How to react to these inputing “events”?
  - Need to write what we call “event handlers”.
- What are the outputs?
  - Display different GUI widgets with proper layout
  - Graphics programming
  - Animation

What Are Controls?

- Example Common Controls:



GUI Programming

Basic Concepts of GUI Programming

- A GUI component is a class that implements the I Component interface.
  - A control, such as a button or label, is a component with a graphical part.
- Some components, which we call containers, can contain other components.
  - Some examples are Form, Panel, Group Box
- Inputs like User actions on components gets translated into events, to which component event handler can respond to.

GUI Programming:
The first class of immediate interest is Component. The Component type provides a canned implementation of the IComponent interface. In the .NET Framework, a component is a class that implements the System.ComponentModel.IComponent interface or that derives directly or indirectly from a class that implements IComponent.
In programming, the term component is generally used for an object that is reusable and can interact with other objects. A .NET Framework component satisfies those general requirements, and additionally provides features such as:

control over external resources
design-time support

GUI Components / Controls

Explanation

- Components and Controls are organized into an inheritance class hierarchy so that they can easily share characteristics.
- Each component / control defines some of the following:
  - properties
  - methods
  - Events

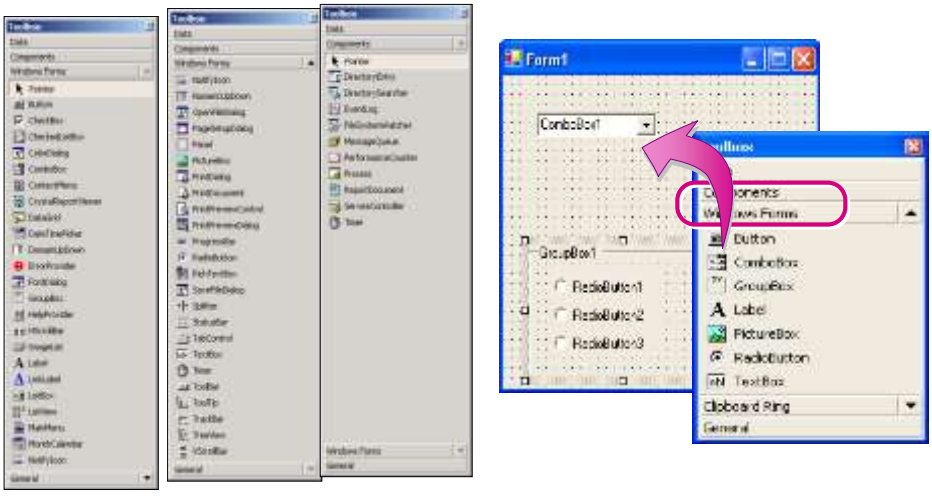
GUI Components / Controls:
The next base class of interest is System.Windows.Forms.Control which establishes the common behaviors required by any GUI-centric type.
A control is a component that provides (or enables) user-interface (UI) capabilities.
The .NET Framework provides two base classes for controls:

one for client-side Windows Forms controls, and
other for ASP.NET server controls
These classes are as follows:
System.Windows.Forms.Control
System.Web.UI.Control.

All controls in the .NET Framework class library derive directly or indirectly from these above two classes.

System.Windows.Forms.Control derives from Component and itself provides UI capabilities.
System.Web.UI.Control implements IComponent and provides the infrastructure on which it is easy to add UI.

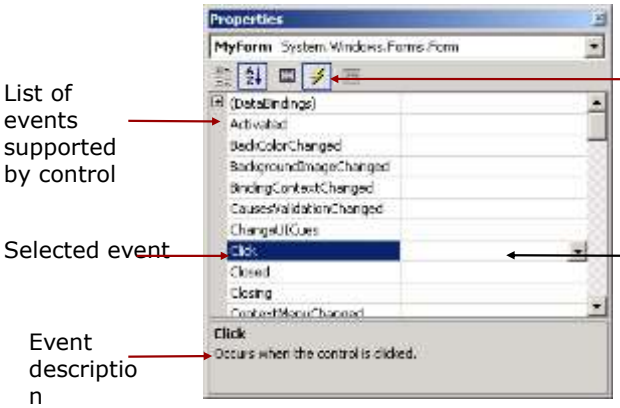
Components and Controls for Windows Forms



GUI Components / Controls (contd.):
The System.Windows.Forms namespace contains a number of types that represent common GUI widgets. Using these types, you can respond to user inputs in a Windows Forms application.

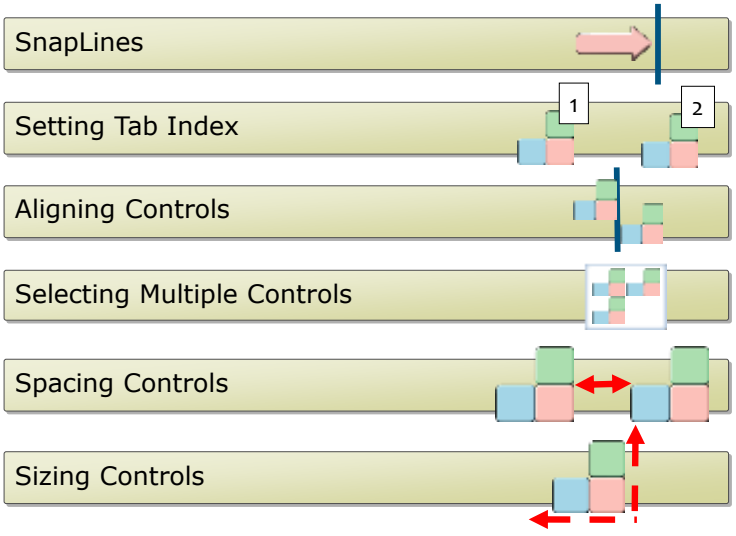
Explanation

- The following figure displays the Events section of the Properties window.



Events section of the **Properties** window.

How to Define the Layout of Controls



How to Add Code to a Control

Creating a default event handler

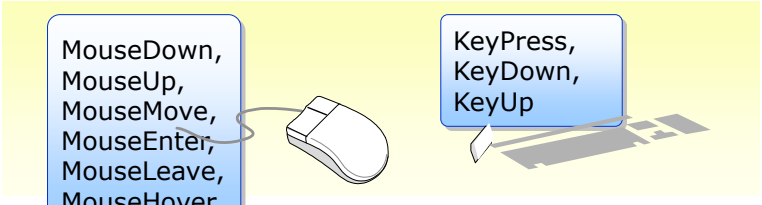
1. Double-click the control.
2. Add code to the automatically generated event handler.

Viewing events in the properties window

1. Click a control.
2. Click the events button in the properties panel menu bar.
3. Scroll through the event list.

What Is an Event?

- Mouse and keyboard

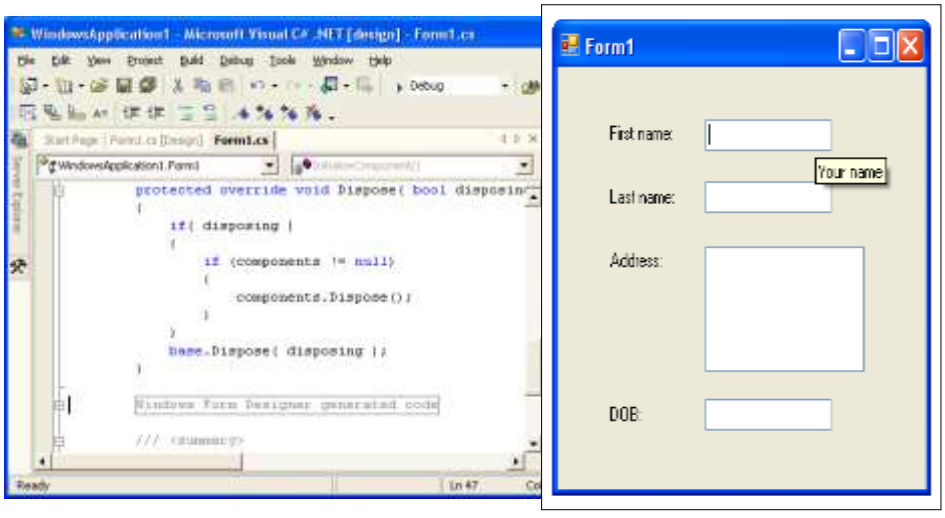


- Property

Event can be either
A user action such as MouseClick or
System/Application generated such as Form_Load

Event Handler is a method that processes an event and perform the tasks.

Windows Forms & Designer-Generated Code



Explanation

- In a control, there is one delegate for each event it can handle.
  - A delegate of an event is an object which references (a list of) methods, that is, event handlers.

Event handler:

It must conform to a given signature.

It must be registered with the delegate object of the event of the control.

Add event handlers to the delegate's invocation list.

Event multicasting:

It can have multiple handlers for one event.

Order called for event handlers is indeterminate.

Example:

```
//Step 1. Class that defines data for the event
//
public class AlarmEventArgs : EventArgs
  {
    private readonly bool snoozePressed = false;
    private readonly int nrings = 0;
    // Constructor.
    public AlarmEventArgs(bool snoozePressed, int nrings) {...}
    // Properties.
    public int NumRings{ get { return nrings;}}
    public bool SnoozePressed { get { return snoozePressed;}}
    public string AlarmText { get {...}}
  }

//Step 2. Delegate declaration.
//
public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);

// Class definition.
//
public class AlarmClock
{
//Step 3. The Alarm event is defined using the event keyword.
//The type of Alarm is AlarmEventHandler.
  public event AlarmEventHandler Alarm;
//
//Step 4. The protected OnAlarm method raises the event by invoking
//the delegates. The sender is always this, the current instance of
//the class.
//
protected virtual void OnAlarm(AlarmEventArgs e)
  {
   if (Alarm != null)
    {
      //Invokes the delegates.
      Alarm(this, e);
    }
  }
}
```

Delegate and Events

- A delegate for the click event of a button. The argument to the constructor contains a reference to the method that performs the event handling logic.
  - EventHandler handler = new EventHandler(button_Click);
  - // Create button, sets their properties, and attaches // an event handler to button.
    - red = new Button();
    - red.Text = "Red";
    - red.Location = new Point(100, 50);
    - red.Size = new Size(50, 50);
    - red.Click +=handler;
    - Controls.Add(red);

Some Common Control Properties

Explanation

- Text property:
  - It specifies the text that appears on a control.
- TabIndex property:
  - It is the order in which controls are given focus.
  - It is automatically set by Visual Studio .NET.
- Enable property:
  - It indicates a control's accessibility.
- Visibility control:
  - It hides control from user.

Labels

Property Description

- Labels provide text instruction.
- It is defined with class Label.
  - It is derived from class Control.

Label Properties	Description / Delegate and Event Arguments
Common Properties	
Font	The font used by the text on the Label.
Text	The text to appear on the Label.
TextAlign	The alignment of the Label's text on the control One of three horizontal positions (left, centre, or right) and one of three vertical positions (top, middle, or bottom)

Labels:
Labels are generally used to provide descriptive text to the user. The text might be related to other controls or the current system state.
You usually see a label together with a text box. The label provides the user with a description of the type of data to be entered in the text box. The Label control is always read-only - the user cannot change the string value of the Text property. However, you can change the Text property in your code.

Text Boxes

Property Description

- TextBoxes provide an area for text input.
- You can specify that a textbox is a password textbox.

Text Box Properties and Events	Description / Delegate and Event Arguments
Common Properties	
AcceptsReturn	If true, pressing ENTER key creates a new line if textbox spans multiple lines. If FALSE, pressing ENTER key clicks the default button of the form.
Multiline	If true, textbox can span multiple lines. Default is false.
PasswordChar	Single character to display instead of typed text, making the TextBox a password box. If no character is specified, Textbox displays the typed text.
ReadOnly	If true, TextBox has a gray background and its text cannot be edited. Default is false.
ScrollBars	For multiline textboxes, indicates which scrollbars appear (none, horizontal, vertical, or both).
Text	The text to be displayed in the text box.
Common Events	(Delegate EventHandler, event arguments EventArgs)
TextChanged	Raised when text changes in TextBox (the user added or deleted characters). Default event when this control is double clicked in the designer.

TextBoxes:
The TextBox control holds text or possibly multiple line of text.
A TextBox control can also be configured as read only and support scroll bars. The immediate base class of TextBox is TextBoxBase, which provides many common behaviors for the TextBox and RichTextBox Controls.

Text Boxes

Buttons

- Buttons are the control to trigger a specific action.
  - They are derived from Button Base.

Button properties and events	Description / Delegate and Event Arguments
Common Properties	
Text	Text displayed on the Button face
Common Events	*(Delegate **EventHandler, event arguments EventArgs**)*
Click	Raised when user clicks the control. Default event when this control is double clicked in the designer.

Buttons:
The role of Button type is to provide a simple vehicle for user input, typically in response to mouse click or key press.
The button class immediately derives from an abstract type named ButtonBase, which provides number of key behaviors for all Button related types (CheckBox, RadioButton, and Button).

Picture Box

Property Description

- Picture Box displays an image.
  - It reads image from file:
    - Image.FromFile() method (need to add System.IO name space)

PictureBox properties and events	Description / Delegate and Event Arguments
Common Properties	
Image	Image to display in PictureBox
SizeMode	Enumeration that controls image sizing and positioning Values **Normal** (default), **StretchImage**, **AutoSize,** and **CenterImage**. **Normal** puts image in top-left corner of **PictureBox**, and **CenterImage** puts image in middle (both cut off image if too large). **StretchImage** resizes image to fit in **PictureBox**. The **AutoSize** resizes PictureBox to hold image.
Common Events	(*Delegate **EventHandler**, event arguments **EventArgs***)
Click	Raised when user clicks the control Default event when this control is double-clicked in the designer

PictureBox:
The PictureBox control is used to display an image. The image can be a BMP, JPEG, GIF, PNG, metafile, or icon. The SizeMode property uses the PictureBoxSizeMode enumeration to determine how the image is sized and positioned in the control. The SizeMode property can be AutoSize, CenterImage, Normal, and StretchImage.
You can change the size of the display of the PictureBox by setting the ClientSize property. You load the PictureBox by first creating an Image-based object.
For example: To load a JPEG file into a PictureBox you will do the following:

```
Bitmap myJpeg = new Bitmap("mypic.jpg"); pictureBox1.Image
= (Image)myJpeg;
```

Notice that you will need to cast back to an Image type because that is what the Image property expects.

Group Boxes

Property Description

- A Group Box can display a caption.
  - The Text property determines its caption.

GroupBox Properties	Description
Common Properties	
Controls	The controls that the GroupBox contains
Text	Text displayed on the top portion of the **GroupBox** (its caption)

Panels

Property Description

- A Panel can have a scrollbar.
  - You can view additional controls inside the Panel.

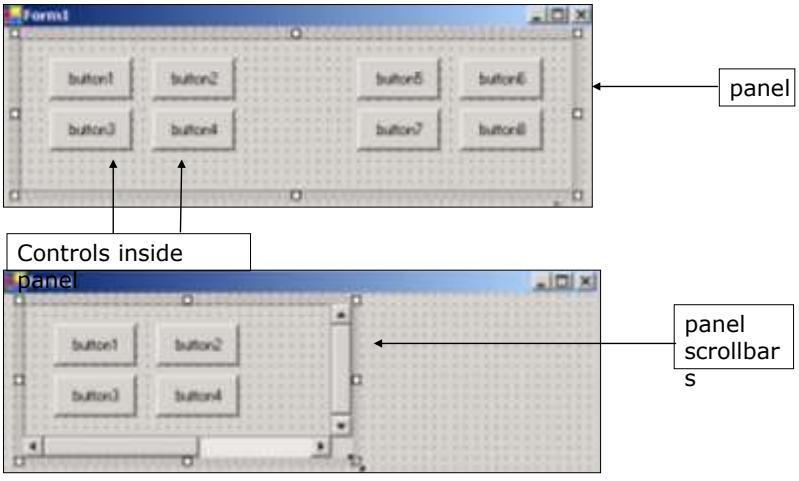
Panel Properties	Description
Common Properties	
AtuoScroll	Whether scrollbars appear when the **Panel** is too small to hold its controls Default is **false.**
BorderStyle	Border of the **Panel** (default None; other options are **Fixed3D** and **FixedSingle**)
Controls	The controls that the **Panel** contains

Panels:
Panel is simply a control that contains other controls. By grouping the controls together and placing them in a panel, it is a little easier to manage the controls.
For example: You can disable all the controls in the panel by disabling the panel.
Since the Panel control is derived from ScrollableControl, you also can get the advantage of the AutoScroll property. If you have too many controls to display in the available area, place them in a Panel and set AutoScroll to true. Now, you can scroll through all of the controls.
Panels do not show a border by default. However, by setting the BorderStyle property to something other than none, you can use the Panel to visually group related controls. This makes the user interface more user-friendly.

Group Boxes and Panels

Illustration

- Let us see an example of creating Panel with scrollbars.



Demo

- Demo with important controls

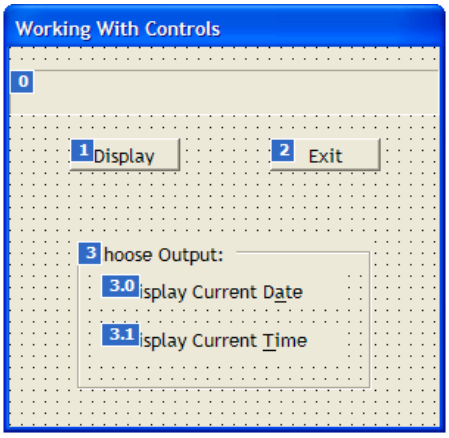
How to Set the Tab Order for Controls

- To set the tab order for controls
  - On the View menu, select Tab Order
  - Click a control to change its tab order

-- OR --

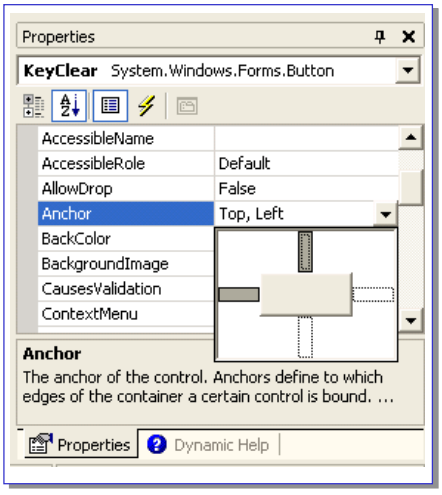
Set the TabIndex property

Set the TabStop property to True



How to Anchor a Control in Windows Forms

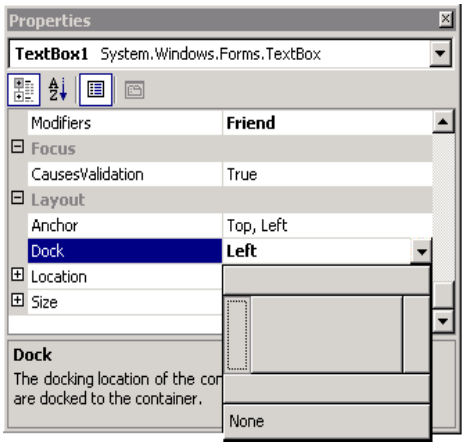
- Anchoring
  - Ensures that the edges of the control remain in the same position with respect to the parent container
- To anchor a control to the form
  - Set its Anchor property
  - Default value: Top, Left
  - Other Styles: Bottom, Right



Set its Anchor Property
Default Value Is Top,Left
Other Styles are Bottom,Right.

How to Dock a Control in Windows Forms

- Docking
  - Enables you to glue the edges of a control to the edges of its parent control
- To dock a control
  - Set the Dock property



You can dock controls to edges of the form.For e.g Windows Explorer docks Treeview control on left side and listview on right side

Select the control you want to dock
In properties window docking contains buttons click the button you want to dock the control .Fill the contents of the control click the Fill button
None- No docking

Dialog Boxes

- Dialogs are a special type of Form used to capture user information or interact with the user in Windows applications
- There are two types of Dialog Boxes available, Modal and Modeless
- These terms relate to how the dialog interacts with the rest of the application
  - Interact with a user to provide or retrieve information
  - Usually a modal style of interaction with the user i.e. Dialog Boxes usually waits for the user response before continuing with the application.

Number of predefined dialog boxes that provide familiar functionality

There exists a set of predefined dialog boxes for capturing common information such as file locations, colors, and printer settings. A custom application will often use custom dialog boxes to facilitate the collection of data from endusers.
When we want to display the dialog box itself, there are two choices: modal or modeless. A modal dialog blocks the current thread and requires the user to respond to the dialog box before continuing with the application. A modeless dialog box is more like a standard window that allows the user to switch the focus back and forth between it and other windows.

Adding Dialog Box

- Three ways of adding dialog boxes are
  - Predefined Dialog Boxes
    - created using InputBox() and MsgBox()
  - Custom Dialog Box
    - Created with forms or by customizing the existing dialog box
  - Standard Dialog Box
    - Created using File / Print / Color / Font Dialogs

Using Msgbox Method

Prompts a message in the dialog box
Waits for the user to respond by clicking a button
Returns an integer indicating the button clicked

Using InputBox Method

Displays a modal dialog box that prompts user to enter some data
Contains Ok and Cancel buttons along with the message to the user

Common Dialogs

- The Dialog boxes available to perform the common functionality such as opening files, selecting fonts, and print previewing
- The .NET Framework provides access to these underlying common dialogs through various classes
- Each of these classes represents a common dialog, and can be displayed as a dialog box

Common Dialog classes -
ColorDialog - This allows a user to select a color from the available color palette as well as define and utilize custom defined colors

FontDialog - This dialog box displays all currently available fonts installed on the system and allows the user to choose one to use in the application

OpenFileDialog - Allows a user to open a file using the standard open file dialog box
SaveFileDialog - This allows a user to select a file, directory or network location to save the application's data to

PrintDialog - This dialog box allows the user to set common page formatting and printing features via the standard print properties dialog box

PrintPreviewDialog - This displays a document as it will appear on the currently-selected printer with the current page settings

Common Dialogs classes

- ColorDialog
  - This allows a user to select a color from the available color palette as well as define and utilize custom defined colors
- FontDialog
  - This dialog box displays all currently available fonts installed on the system and allows the user to choose one to use in the application

Example of Color Dialog Box

```
private void button1_Click(object sender, System.EventArgs e)
 {
   ColorDialog MyDialog = new ColorDialog();
   // Keeps the user from selecting a custom color.
   MyDialog.AllowFullOpen = false ;
   // Allows the user to get help. (The default is false.)
   MyDialog.ShowHelp = true ;
   // Sets the initial color select to the current text color.
   MyDialog.Color = textBox1.ForeColor ;

   // Update the text box color if the user clicks OK
   if (MyDialog.ShowDialog() == DialogResult.OK)
      textBox1.ForeColor =  MyDialog.Color;
 }
```

Example of FontDialog

```
private void button1_Click(object sender, System.EventArgs e)
 {
   fontDialog1.ShowColor = true;

   fontDialog1.Font = textBox1.Font;
   fontDialog1.Color = textBox1.ForeColor;

   if(fontDialog1.ShowDialog() != DialogResult.Cancel )
   {
     textBox1.Font = fontDialog1.Font ;
     textBox1.ForeColor = fontDialog1.Color;
   }
 }
```

Demo

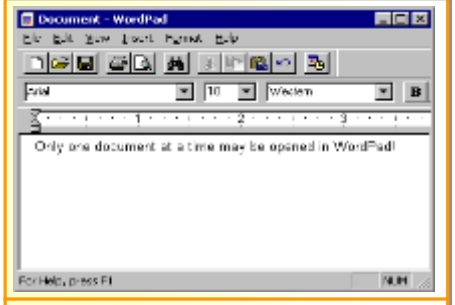
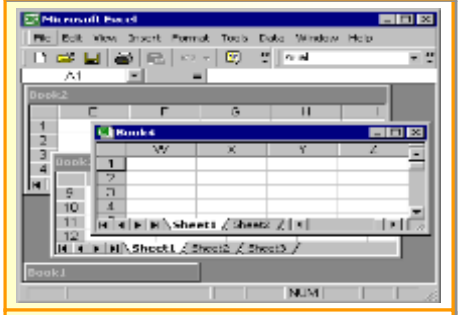
- Demo on Dialog Boxes

MDI Applications

Explanation

- Multiple Document Interface (MDI) applications have a single, primary window (the parent window) that contains a set of windows (the child windows) within its client region.
  - Each child window is a form that is constrained to appear only within the parent.
  - Children typically share the menu bar, tool bar, and other parts of the parent's interface.
  - Secondary windows like dialog boxes are not constrained to the parent window's client region.

SDI vs. MDI Applications

SDI	MDI
	
Only one document is visible	Displays multiple documents at the same time
You must close one document before you open another	Each document is displayed in its own window

Creating MDI Applications

Explanation

- Creating MDI Parent Form:
  - The base of a Multiple Document Interface (MDI) is the MDI parent form.
  - This is the form that holds the MDI child windows, which are all the "sub-windows" through which the user interacts with the MDI application.
  - To create MDI Parent Form, create a new form, and add the code.

```
this.IsMDIContainer = true;
```

Creating MDI Applications:

You can create an MDI application by using the following steps:

Create a Form (MainForm) that represents the MDI parent window.
Set its IsMdiContainer property to true. The following code demonstrates how to set this property.

```
this.IsMdiContainer = true;
```

Explanation

- Creating MDI Child Forms:
  - A vital constituent of Multiple Document Interface (MDI) applications is the MDI child forms. These are the main windows for client interaction.
  - To Create MDI Child Forms:

```
Form frmchild=new Form();
  frmchild.MDIParent=this;
  frmchild.Show();
```

Creating MDI Applications – Creating MDI Child Forms:
Create child forms and set the MdiParent property of each form to reference the parent form. The following code demonstrates setting the MDI parent for an instance of a child form.

```
Form frmchild=new Form();
  frmchild.MDIParent=this;
```

If you have different types of data to display, you can have multiple types of child forms. To display a child form, create an instance of the child form and call its Show method.

```
frmchild.Show();
```

Explanation

- Determining the Active MDI Child:
  - Use the ActiveForm property of an MDI Form, which returns the child form that has the focus.

- Arranging Child Forms:
  - Use the LayoutMDI method with the MDILayout enumeration to rearrange the child forms in an MDI parent form.
    - ArrangeIcons
    - Cascade
    - TileHorizontal
    - TileVertical

Creating MDI Applications – Creating MDI Child Forms:
Determining the Active MDI Child:
In a number of circumstance, we desire to give a command that operates on the control with the focus on the current active child form.
For the reason that an application can have many instances of the same child form, the process wants to be acquainted with which form to use. To specify this, use the ActiveForm property of an MDI Form, which returns the child form that has the focus.
In any case, one MDI child form must be loaded and visible when you access the ActiveForm property, or an error is returned.
Arranging Child Forms:
Frequently, applications will have menu commands for actions such as Tile, Cascade, and Arrange, which concern to the open MDI child forms. One can use the LayoutMDI method with the MDILayout enumeration to rearrange the child forms in an MDI parent form.
The MDILayout enumeration can be set to four different values, which will display child forms as cascading, either horizontally or vertically. Often, these methods are used as the event handlers called by a menu item's Click event. In this way, a menu item with the text “Cascade Windows” can have the desired effect on the MDI child windows.

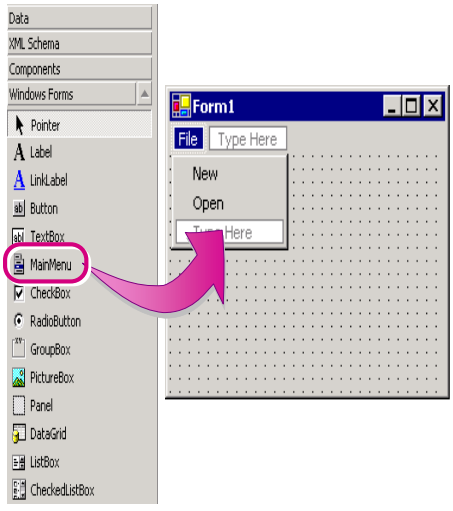
Demo

- Demo on creating MDI application

Menus

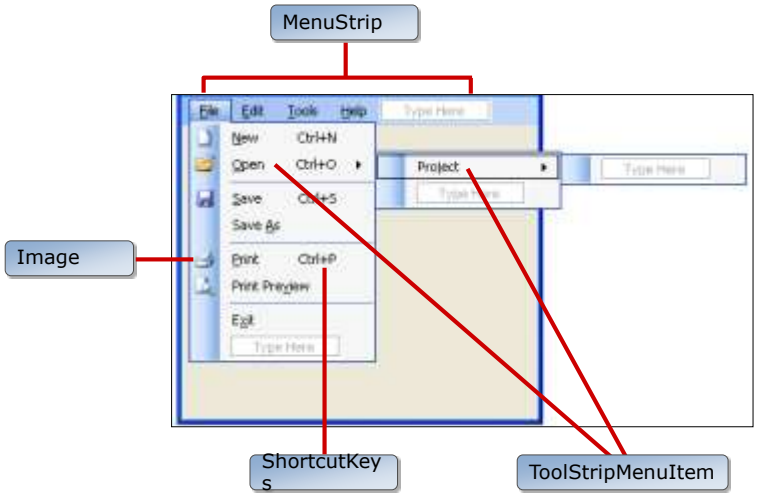
Explanation

- A menu on a form is created with a MainMenu object, which is a collection of MenuItem objects.
- You can add menus to Windows Forms at design time by adding the MainMenu control, and then add menu items to it using the Menu Designer.
- Menus can also be added programmatically by adding one or more MainMenu controls to a form and adding MenuItem objects to the collection.

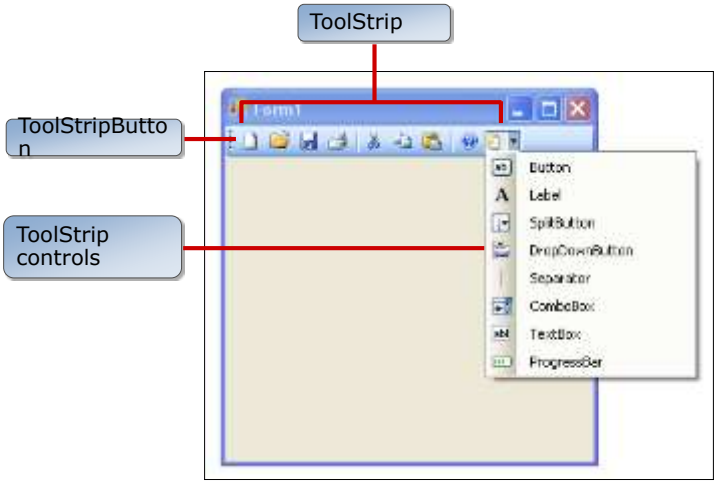


Menus:
The main window of nearly all Windows applications includes a menu across the top. This is called the main menu. The main menu typically contains top-level categories, such as File, Edit, and Tools. From the main menu descend drop-down menus, which contain the actual selections associated with the categories. When a menu item is selected, a message is generated. Therefore, to process a menu selection, your program will assign an event handler to each menu item.
Prior to C# 2.0, there was only one way to create a main menu: that is by using the classes MainMenu and MenuItem, both of which inherit the Menu class. These classes create the traditional-style menus that have been used in Windows applications for years. C# 2.0 still supports these classes and traditional-style menus. However, it adds a second way to add menus to a window by using a new set of classes based on ToolStrip, such as MenuStrip and ToolStripMenuItem. Menus based on ToolStrip have many additional capabilities and give a modern look and feel.

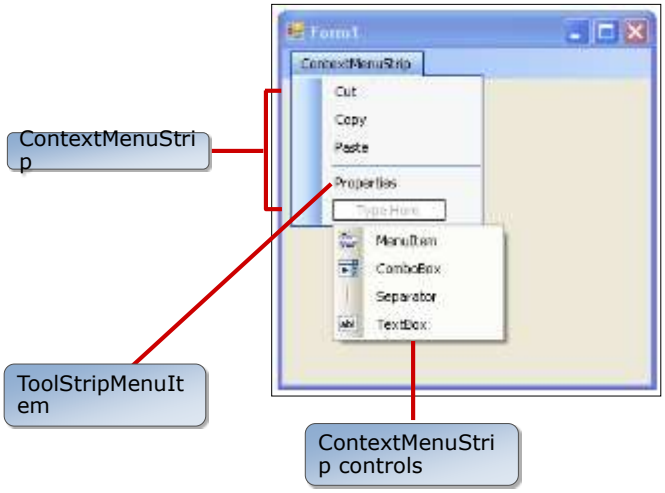
How to Create a Menu



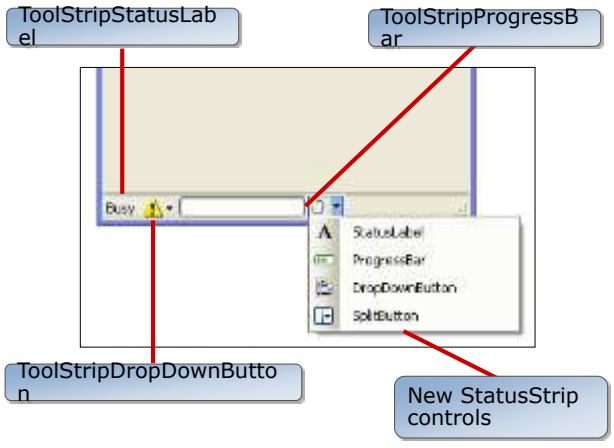
How to Create a Tool Strip



How to Create a Context Menu



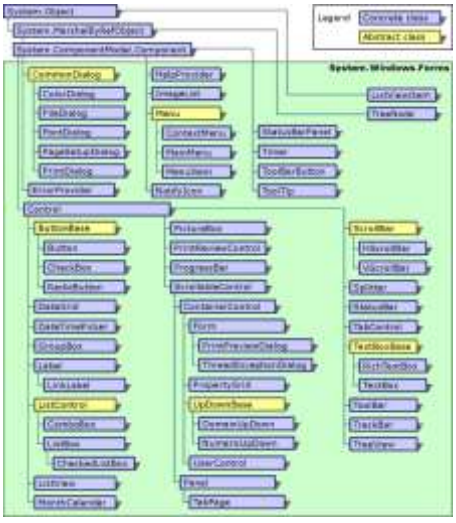
How to Create a Status Bar



Demo

- Demo on creating Menus,ToolStrip…

Windows Forms Hierarchy



Summary

- In this lesson, you have learnt:
  - Differences between Console Application and Windows Application
  - Delegate based Event Handling Model in .NET
  - Different Controls in .NET and their properties
  - What is Anchoring and Docking?
  - How do you create MDI Application in C# ?

Summary

- In this lesson, you have learnt (contd.):
  - How do you create Menus in C# ?

Review Question

- Question 1: What are the challenges for a Windows application?
- Question 2: Can I have two event handlers for one event? How?
- Question 3: Can two events point to the same event Handler? How?
- Question 4: What is Anchoring and Docking?
- Question 5: What are the different types of Menus you can create using C#?