UNIT 2
Windows Application

(6 Hours) — Advanced .NET Technology (CACS408)

Note: This PDF may contain errors — please cross-check important points with your syllabus, prescribed textbook and teacher's notes before the exam.

Note: The syllabus says "Window Forms"; the correct name is "Windows Forms" (WinForms). Windows Forms runs only on Windows, both on .NET Framework and on modern .NET.

2.1 Windows Forms

Windows Forms (WinForms) is a GUI framework in the `System.Windows.Forms` namespace for building desktop applications. A **Form** is a window; **controls** (buttons, text boxes, etc.) are placed on it using the Visual Studio designer, and code responds to **events**.

hcalink.com

Benefits of Windows Forms

- **Rapid Application Development (RAD):** drag-and-drop designer and a rich toolbox of ready-made controls.
- **Event-driven model** that is simple to learn and code.
- **Rich native look and feel** with full access to the Windows OS, files, printers and hardware.
- **Data binding** to databases, objects and XML with little code.
- **Easy deployment** (ClickOnce, MSI/installer) and strong support for custom and third-party controls.
- Full use of the .NET class library and GDI+ for custom drawing.

Structure of a WinForms program

```
using System;
using System.Windows.Forms;

namespace HelloApp
{
    public partial class Form1 : Form          // Form1.cs (your code)
    {
        public Form1() { InitializeComponent(); }

        private void btnShow_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Hello, " + txtName.Text);
        }
    }

    static class Program                        // Program.cs (entry point)
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());       // starts the message loop
        }
    }
}
```

Program 2.1: Minimal Windows Forms application

The designer writes control declarations and property settings into **Form1.Designer.cs** (a partial class, method `InitializeComponent()`), so your own logic stays in **Form1.cs**.

2.2 Windows Forms Controls, Properties and Events

Control	Use	Important members
Label	Shows read-only text.	Text, Font, ForeColor
TextBox	Single/multi-line text input.	Text, MaxLength, PasswordChar, Multiline, TextChanged
Button	Triggers an action.	Text, Enabled, Click
CheckBox / RadioButton	Yes/no or one-of-many choice.	Checked, CheckedChanged
ListBox / ComboBox	List of items (combo = drop-down).	Items, SelectedItem, SelectedIndex, SelectedIndexChanged
DateTimePicker	Date selection.	Value, Format
PictureBox	Displays an image.	Image, SizeMode
Panel / GroupBox / TabControl	Containers that group controls.	Controls, Dock, TabPages
DataGridView	Tabular data display/edit.	DataSource, Rows, Columns
Timer	Fires an event at intervals.	Interval, Enabled, Tick
MenuStrip / ToolStrip / StatusStrip	Menus, toolbars, status bar.	Items

bealink.com

Common properties and events

Common Properties	Common Events
`Name`, `Text`, `Size`, `Location`, `Font`, `BackColor`, `ForeColor`, `Enabled`, `Visible`, `Dock`, `Anchor`, `TabIndex`, `Tag`	`Click`, `DoubleClick`, `TextChanged`, `KeyPress`, `KeyDown`, `MouseEnter`, `MouseMove`, `Load`, `FormClosing`, `Resize`, `Paint`

2.3 .NET Events and Handling Multiple Events

In .NET, an **event** is a message sent by an object (the **publisher**) to signal that something happened; other objects (**subscribers**) register **event handlers**. Events are built on **delegates** — type-safe references to methods. The standard handler signature is `void Handler(object sender, EventArgs e)`, where *sender* is the control that raised the event and *e* carries event data.

```
// Publisher with a custom event
public class Alarm
{
    public event EventHandler Ring;                 // event declaration
    public void Trigger() { Ring?.Invoke(this, EventArgs.Empty); }
}

// Subscriber
Alarm al = new Alarm();
al.Ring += (s, e) => MessageBox.Show("Wake up!");   // subscribe
al.Trigger();

// Wiring a control event in code
btnOk.Click += new EventHandler(btnOk_Click);
```

Program 2.2: Delegate-based event

Handling multiple events with one handler

One method can handle events of several controls; use *sender* to find out which control fired. Likewise, one control can have several handlers (they run in the order they were added).

bcalink.com

```
public Form1()
{
    InitializeComponent();
    btnRed.Click   += ColorButton_Click;     // same handler for 3 buttons
    btnGreen.Click += ColorButton_Click;
    btnBlue.Click  += ColorButton_Click;
}
private void ColorButton_Click(object sender, EventArgs e)
{
    Button b = (Button)sender;               // which button was clicked?
    this.BackColor = b.BackColor;
    lblInfo.Text = b.Text + " selected";
}
```

Program 2.3: One handler for multiple controls

2.4 MDI Forms and Form Inheritance

A **Multiple Document Interface (MDI)** application has one parent window containing several child windows (like old Word/Excel). The parent sets `IsMdiContainer = true`; each child sets `MdiParent`.

```
// Parent form: IsMdiContainer = true (property window or code)
private void menuNew_Click(object sender, EventArgs e)
{
    ChildForm child = new ChildForm();
    child.MdiParent = this;          // make it a child
    child.Show();
}
private void menuCascade_Click(object s, EventArgs e)
{ LayoutMdi(MdiLayout.Cascade); }
private void menuTile_Click(object s, EventArgs e)
{ LayoutMdi(MdiLayout.TileHorizontal); }
// other: MdiLayout.TileVertical, ArrangeIcons
// this.ActiveMdiChild -> currently active child; this.MdiChildren -> all children
```

Program 2.4: MDI parent and child

bcslink.com

Form inheritance (visual inheritance)

A new form can **inherit** from an existing form and automatically get its controls, layout and code. This gives a consistent look (e.g., a company logo and standard buttons). Controls in the base form should be `protected` or `public` (Modifiers property) to be editable in the derived form.

```
public class BaseForm : Form              // base form with common controls
{
    protected Button btnClose = new Button { Text = "Close" };
    public BaseForm()
    {
        btnClose.Click += (s, e) => Close();
        Controls.Add(btnClose);
    }
}
public class CustomerForm : BaseForm      // inherits everything
{
    public CustomerForm() { Text = "Customer Entry"; }
}
```

Program 2.5: Form inheritance

2.5 Dialogs, Tooltips and Resizing

Dialog boxes

A **modal** dialog (`ShowDialog()`) must be closed before the user can use the parent form; a **modeless** dialog (`Show()`) allows switching back. `ShowDialog()` returns a `DialogResult` (OK, Cancel, Yes, No…).

Dialog	Purpose
MessageBox	Shows a message with buttons/icon: `MessageBox.Show(text, caption, buttons, icon)`
OpenFileDialog / SaveFileDialog	Choose a file to open / a file name to save (Filter, FileName).
FolderBrowserDialog	Select a folder.
ColorDialog / FontDialog	Choose a color / font.
PrintDialog, PrintPreviewDialog	Printer selection and preview.
Custom dialog	Your own Form shown with ShowDialog(); set AcceptButton, CancelButton and DialogResult.

```
DialogResult r = MessageBox.Show("Delete record?", "Confirm",
                 MessageBoxButtons.YesNo, MessageBoxIcon.Question);
if (r == DialogResult.Yes) { /* delete */ }

OpenFileDialog ofd = new OpenFileDialog();
ofd.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
if (ofd.ShowDialog() == DialogResult.OK)
    txtBox.Text = File.ReadAllText(ofd.FileName);

ColorDialog cd = new ColorDialog();
if (cd.ShowDialog() == DialogResult.OK) this.BackColor = cd.Color;
```

Program 2.6: Message box and common dialogs

bcalink.com

Tooltips

The `ToolTip` component shows a small hint when the mouse hovers over a control.

```
ToolTip tip = new ToolTip();
tip.SetToolTip(btnSave, "Save the current record");
tip.AutoPopDelay = 5000;     // ms the tip stays visible
```

Program 2.7: ToolTip

Resizing and layout

- **Anchor**: keeps a control a fixed distance from chosen edges (Top, Bottom, Left, Right) when the form is resized.
- **Dock**: attaches a control to an edge (Top, Fill, etc.) so it stretches automatically.
- **MinimumSize / MaximumSize** limit resizing; the **Resize / SizeChanged** events allow custom code.
- Layout containers: **TableLayoutPanel**, **FlowLayoutPanel**, **SplitContainer**, **Panel**.
- **AutoScroll** adds scroll bars when content does not fit.

2.6 Menus and Context Menus

MenuStrip creates the main menu bar made of `ToolStripMenuItem` objects. A **ContextMenuStrip** is a pop-up (right-click) menu attached to a control through its `ContextMenuStrip` property.

```
MenuStrip ms = new MenuStrip();
ToolStripMenuItem file = new ToolStripMenuItem("&File");     // Alt+F
ToolStripMenuItem exit = new ToolStripMenuItem("E&xit");
exit.ShortcutKeys = Keys.Control | Keys.Q;
exit.Click += (s, e) => Application.Exit();
file.DropDownItems.Add(exit);
ms.Items.Add(file);
this.MainMenuStrip = ms;  this.Controls.Add(ms);

// Context menu for a text box
ContextMenuStrip cms = new ContextMenuStrip();
cms.Items.Add("Copy", null, (s, e) => txtBox.Copy());
cms.Items.Add("Paste", null, (s, e) => txtBox.Paste());
txtBox.ContextMenuStrip = cms;
```

Program 2.8: Menu and context menu

2.7 Custom Control Creation

When the built-in controls are not enough, you can create your own reusable controls. There are three approaches:

localink.com

Type	How	Example
Extended control	Inherit from an existing control (e.g., TextBox) and add/override behavior.	NumericOnlyTextBox
User control (composite)	Inherit from UserControl; combine several controls in one reusable unit.	Address panel = labels + textboxes
Custom (owner-drawn) control	Inherit from Control and override OnPaint() to draw everything yourself.	Gauge, rating stars

```
public class NumericTextBox : TextBox               // extended control
{
    protected override void OnKeyPress(KeyPressEventArgs e)
    {
        if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar))
            e.Handled = true;                       // block non-digits
        base.OnKeyPress(e);
    }
}

public class ColorLabel : Control                   // owner-drawn control
{
    public Color Fill { get; set; } = Color.Orange;
    public event EventHandler FillChanged;          // custom event
    protected override void OnPaint(PaintEventArgs e)
    {
        e.Graphics.FillEllipse(new SolidBrush(Fill), ClientRectangle);
        base.OnPaint(e);
    }
}
```

Program 2.9: Extended and owner-drawn controls

After building the project, the control appears in the Toolbox and can be dropped on forms like any other.

bcalink.com

2.8 Graphics and GDI+

GDI+ (Graphics Device Interface Plus) is the 2-D drawing API in `System.Drawing`. You draw on a `Graphics` object, usually obtained from the `Paint` event (`e.Graphics`) or `CreateGraphics()`. The origin (0,0) is the top-left corner of the client area; X grows to the right and Y downward, measured in pixels.

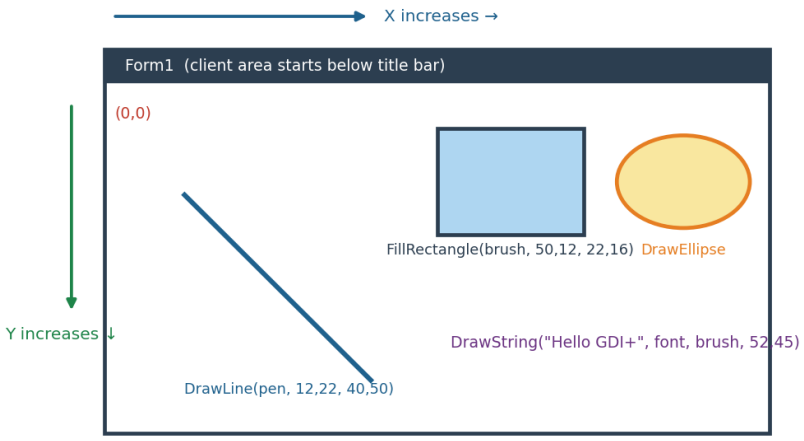


Fig 2.1: GDI+ coordinate system and drawing methods

Object / Method	Purpose
Pen	Draws outlines (color, width, DashStyle): DrawLine, DrawRectangle, DrawEllipse, DrawArc, DrawPolygon, DrawCurve.
Brush	Fills shapes: SolidBrush, HatchBrush, LinearGradientBrush, TextureBrush → FillRectangle, FillEllipse, FillPie.
Font / DrawString	Draws text at a position.
DrawImage	Draws a Bitmap/Image.
Color, Point, Rectangle	Helper structures.

healink.com

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    using (Pen pen = new Pen(Color.Blue, 3))
        g.DrawLine(pen, 12, 22, 120, 90);
    using (SolidBrush br = new SolidBrush(Color.LightSkyBlue))
        g.FillRectangle(br, 150, 20, 100, 60);
    g.DrawEllipse(Pens.Red, 270, 20, 80, 60);
    g.DrawString("Hello GDI+", new Font("Arial", 16), Brushes.Purple, 20, 120);
    // g.DrawImage(Image.FromFile("logo.png"), 20, 160);
}
// call this.Invalidate() to force a repaint
```

Program 2.10: Drawing in the Paint event

Note: Always dispose Pens, Brushes and Graphics created by you (use a using block). Draw in the Paint event so the drawing is not lost when the window is minimized or covered.

Key Takeaways — Unit 2

✓ Windows Forms = desktop GUI framework (System.Windows.Forms); designer code lives in Form.Designer.cs.

✓ Controls have properties (Text, Size, Enabled…) and events (Click, TextChanged, Load…).

✓ Events use delegates: publisher raises, subscribers handle; one handler can serve many controls via sender.

✓ MDI: parent IsMdiContainer = true, child MdiParent = this; LayoutMdi arranges children.

✓ Form inheritance reuses a base form; dialogs are modal (ShowDialog) or modeless (Show).

✓ Resize with Anchor, Dock and layout panels; MenuStrip and ContextMenuStrip create menus.

✓ Custom controls: extend an existing control, build a UserControl, or override OnPaint.

✓ GDI+ draws with Graphics + Pen/Brush in the Paint event; origin is top-left.

— End of Unit 2 —

bcalink.com