Windows Forms

.NET Framework

- Microsoft technology introduced in 2002
  - 1.1 – 2003, 2.0 – 2005, 3.0 – 2006, 3.5 – 2007
- Integration with Windows systems:
  - .NET Framework 1.0, 1.1, 2.0:
    - available for: 98, NT 4.0, 2000+
    - ver. 1.1 included in Windows Server 2003
    - Microsoft .NET Framework Version 2.0 Redistributable Package (x86) – 22.4 MB for installer, 280 MB disk space
  - .NET Framework 3.0, 3.5:
    - available for: XP SP2, 2003, Vista, 2008
    - ver. 3.0 included in Windows Vista and Windows Server 2008
    - ver. 3.5 – 197 MB full installer, up to 500 MB disk space

Windows Forms (WinForms)

- The classes within the Microsoft .NET Framework that are used to create traditional Microsoft Windows GUI desktop applications
  - the **`System.Windows.Forms`** namespace
- Categories of classes:
  - core infrastructure (e.g. **`Application`**, **`Form`**)
  - controls – derived from the **`Control`** class (e.g. **`Button`**, **`TextBox`**)
  - components – not derived from the **`Control`** class (e.g. **`Timer`**, **`ToolTip`**)
  - common dialog boxes (e.g. **`OpenFileDialog`**, **`PrintDialog`**)

Sample Program

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

namespace MyWindowsApp
{
    public class MainWindow : Form
    {
        static void Main(string[] args)
        {
            Application.Run(new MainWindow());

            MessageBox.Show("That's all",
                            "Sample program",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Stop);
        }
    }
}
```

Event-Handling Model

- The graphical user interface is event-driven – it generates events in response for the user's interaction
- Event handlers - methods that process events and perform tasks
- Each control generating an event has an associated delegate that defines the signature for event handlers
  - event delegates are multicast (they contain lists of method references)
  - once an event is raised, every method that the delegate references is called

Using Events

```
namespace MyApplication {
  public class MyForm : Form {
    public MyForm() {
      FormClosing += new FormClosingEventHandler(OnClosing);
    }

    private void OnClosing(Object sender,
                           FormClosingEventArgs e) {
      if (MessageBox.Show("Sure to close?", "Question",
                MessageBoxButtons.YesNo) == DialogResult.No) {
        e.Cancel = true;
      }
    }
  }
}
```

```
namespace MyApplication {
  static class Program {
    static void Main() {
      Application.ApplicationExit += new
          EventHandler(Program_ApplicationExit);
      Application.Run(new MyForm());
    }

    private static void Program_ApplicationExit(
                Object sender, EventArgs e) {
      MessageBox.Show("That's all");
    }
  }
}
```

The `Application` Class

- It provides static methods and properties to manage an application
  - **`Run()`**
  - **`Exit()`, `ExitThread()`**
  - **`DoEvents()`** – very useful during long-lasting calculations to handle any pending input
  - **`EnableVisualStyles()`** – supporting Windows XP visual styles
  - **`Idle`** event – fired whenever some input has just been handled and the application is about to become idle
  - **`ApplicationExit`** event – notification when the application is exiting
  - properties: **`ExecutablePath`**, **`StartupPath`**

The `Form` Class

- The class representing a main window, dialog box, or MDI child window
- Inheritance hierarchy:

```
Object
  MarshalByRefObject
    Component
      Control
        ScrollableControl
          ContainerControl
            Form
```

Form's Lifetime

1. `Constructor`
   - `InitializeComponent()` is called to create and initialize all child controls added using the Form Designer (in Visual Studio)
2. `Form.Load`
3. `Form.Activated`
4. `//...`
5. `Form.Deactivate`
6. `Form.FormClosing`
7. `Form.FormClosed`
8. `Dispose` (a part of the `IDisposable` interface)
   - use it for disposing any resources used by the form
9. `Destructor` (the Garbage Collector will call it)

Form's Size and Position

- Visibility
  - `Show()`
  - `Visible`
  - `Shown, VisibleChanged`
- Properties:
  - `Region, Bounds, DesktopBounds, ClientRectangle`
  - `Left, Top, Right, Bottom`
  - `StartPosition,`
  - `Location, DesktopLocation`
  - `Width, Height,`
  - `Size, ClientSize,`
  - `MinimumSize, MaximumSize`
  - `AutoSize, AutoSizeMode`
  - `WindowState, TopMost`

Form's Size and Position cont.

- Methods:
  - `SetBounds()`, `SetDesktopBounds()`, `SetDesktopLocation()`
  - `CenterToParent()`, `CenterToScreen()`
  - `BringToFront()`
  - `PointToClient()`, `PointToScreen()`, `RectangleToClient()`, `RectangleToScreen()`
  - `SizeFromClientSize()`
- Events:
  - `ClientSizeChanged`, `SizeChanged`
  - `LocationChanged`
  - `MaximumSizeChanged`, `MinimumSizeChanged`
  - `Resize`, `ResizeBegin`, `ResizeEnd`

Form's Appearance

- Properties:
  - ForeColor, BackColor
  - BackgroundImage, BackgroundImageLayout (Center, None, Stretch, Tile, Zoom)
  - Cursor, Icon
  - Font
  - FormBorderStyle, ShowIcon, ControlBox, MinimizeBox, MaximizeBox, HelpButton, SizeGripStyle
  - ShowInTaskbar, TopLevel
  - Opacity
  - TransparencyKey

Form's Appearance cont.

- Methods:
  - `ResetBackColor(), ResetForeColor(),`
  - `ResetFont(), ResetCursor()`
- Events:
  - `ForeColorChanged, BackColorChanged`
  - `BackgroundImageChanged, BackgroundImageLayoutChanged`
  - `FontChanged`

The Mouse

- Mouse events
  - `MouseMove, MouseDown, MouseUp, MouseClick, MouseDoubleClick`
    - `MouseEventArgs {Button, Clicks, Delta, Location, X, Y}`
  - `MouseEnter, MouseHover, MouseLeave, MouseWheel`
  - `Click, DoubleClick`
- The `Cursor` class
  - the contructor loads the cursor from a stream, file, program, or the system's resources
  - the `Cursor` property of a form
  - for standard cursors use the `Cursors` class, e.g. `Cursors.WaitCursor`

The Keyboard

- Focus
  - **`CanFocus`**, **`Focused`**, **`ContainsFocus`** (including children)
  - **`Focus()`**
  - **`GotFocus`**, **`LostFocus`** (low level events)
  - **`Enter`**, **`Leave`** (for controls), **`Activated`**, **`Deactivate`** (for forms)
- Events
  1. **`KeyDown (KeyEventArgs: KeyCode, KeyData, Modifiers, Alt, Control, Shift)`**
  2. **`KeyPress (KeyPressEventArgs: KeyChar)`**
  3. **`KeyUp (KeyEventArgs)`**
- **`Keys`** enumeration (e.g. **`Keys.Q`**, **`Keys.F5`**, **`Keys.LShiftKey`**)

Modal and Modeless Forms

MODAL

- Blocks input for any other windows of the application
- `ShowModal()`
- `Close()`
  - `FormClosing`
  - `FormClosed`
- `CancelButton`, `AcceptButton`
- `DialogResult`

MODELESS

- `Show()`
- `Close()`
  - `FormClosing`
  - `FormClosed`

Handling Win32 Messages

```
private const int HTCAPTION = 0x0002;
private const int WM_NCHITTEST = 0x0084;


protected override void WndProc(ref Message m)
{
  switch (m.Msg)
  {
    case WM_NCHITTEST:
      m.Result = (IntPtr)HTCAPTION;
      break;
    default:
      base.WndProc (ref m);
      break;
  }
}
```

- `Message: {HWnd, LParam, Msg, Result, WParam}`
- `Form.Handle` – `HWND` as the `IntPtr` type

Application Settings

- .config files
  - can be read and modified by applications
  - XML
- Dynamic properties

```
<configuration>
  <appSettings>
    <add key="welcome" value="Hello my user"/>
    <add key="Form1.Opacity" value="1"/>
  </appSettings>
</configuration>
```

```
AppSettingsReader reader = new AppSettingsReader();
string s = (string)reader.GetValue(
    "welcome", typeof(string));
```

Resources

- An assembly is a collection of types and optional resources
  - the binary data, text files, audio files, video files, string tables, icons, images, XML files
- Localized applications
  - a problem with multilingual user interface
  - for each resource added to an assembly, it is possible to specify the culture information (a language and country, e.g. "pl-PL", "en-US", "de-De", "de-AT")
  - satellite assemblies

Resource Files

```
Language=Polish
Next=Następna strona
Prev=Poprzednia strona
```

- .txt
  - textual name/value format
  - an easy way to add string resources
- .resx
  - the XML format
  - support for both strings and other objects such as images
- .resources
  - the binary format
  - a binary equivalent of the XML file
  - the only format that can be embedded in an assembly, the other formats must be converted

Creating .resx Files

```
ResXResourceWriter w =
   new ResXResourceWriter(@"C:\myRes.resx");

Image img = new Bitmap("pattern.bmp");
w.AddResource("background", img);

w.AddResource("Next", "Następny");

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

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  (...)
  <data name="background"
        type="System.Drawing.Bitmap, System.Drawing"
        mimetype="application/x-microsoft.net.object.bytearray.base64">
    <value>
        Qk32BAAAAAAAAHYAAAAoAAAAMAAAADAAAAABAAQ
        AAAAAAAAAAADDDgAAww4AABAAAAAQAAAAAAA/w
        ...
    </value>
```

Creating .resources Files

- Conversion from .resx and .txt files

```
resgen myRes.resx myRes.resources
resgen polish.txt polish.resources
```

- Using the source code

```
ResourceWriter rw = new
         ResourceWriter(@"C:\myRes.resources");

Image img = new Bitmap("pattern.bmp");
w.AddResource("background", img);

w.AddResource("HelloWorld", "Witaj świecie");

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

Adding the .resources File to an Assembly

- Resources must by embedded into a .NET assembly
  - a manifest describing assembly's will be updated

```
csc /t:exe /resource:polish.resources
           /resource:myRes.resources myApp.cs
```

Programmatically Reading Resources

```
ResourceManager rm = new ResourceManager(
    "myApp.myRes",
    Assembly.GetExecutingAssembly());

MessageBox.Show(rm.GetString("HelloWorld"));

pictureBox.Image =
    (Bitmap)rm.GetObject("background");

rm.ReleaseAllResources();
```

Using Visual Studio

- Adding resources
  - menu: Project / Add New Item / Resources File
  - resources can be linked or embedded
  - a property in a special class will be generated (Resources1.Designer.cs)
- Editting resources
  - Visual Studio built-in editors
    - the binary editor, image editor
  - external editors (e.g. the Paint for image files)
    - other applications can be associated with types of resources
- Compiling resources into assemblies
  - the resgen.exe tool is called

Linked vs. Embedded Resources

- Linked resources (the default)
  - stored as files within the project
  - there is only a relative path or a link to a file on disk in .resx file
  - during compilation, the resource data is taken from the files and placed into the manifest for the application
- Embedded resources
  - the resource data is stored directly in the .resx file (in a text representation of the binary data)

- In both cases, the resource data is compiled into the executable file

Using the Source Code Generated by VS

```
internal class Resource1 {
  internal static string String1 {
    get {
      return ResourceManager.GetString(
                        "String1", resourceCulture);
    }
  }
  internal static System.Drawing.Bitmap FeatherTexture {
    get {
      object obj = ResourceManager.GetObject(
                        "FeatherTexture", resourceCulture);
      return ((System.Drawing.Bitmap)(obj));
    }
  }
}
```

```
pictureBox1.Image = Resource1.FeatherTexture;
label1.Text = Resource1.String1;
```