Visual C# .Net using framework 4.5

Eng. Mahmoud Ouf

Lecture 09

Multiple-Document-Interface (MDI) Windows

Till now, we have built only single-document-interface (SDI) applications. Such programs (including Notepad or Paint) support only one open window or document at a time.

To edit multiple documents, the user must create additional instances of the SDI application.

Multiple document interface (MDI) programs (such as Adobe Photoshop) enable users to edit multiple documents at once. MDI programs also tend to be more complex.

The application window of an MDI program is called the parent window, and each window inside the application is referred to as a child window.

Although an MDI application can have many child windows, each has only one parent window.

Multiple-Document-Interface (MDI) Windows

Note:

Only one child window can be active at a time.

Child windows cannot be parents themselves and cannot be moved outside their parent.

A child window's functionality can be different from the functionality of other child windows of the parent.

For example, one child window might edit images, another might edit text and a third might display network traffic graphically, but all could belong to the same MDI parent.

Multiple-Document-Interface (MDI) Windows

To create an MDI form:

1) create a new Form and set its IsMDIContainer property to True
2) create a child form class to be added to the form. (To do this, right-click the project in the Solution Explorer, select Add Windows Form... and name the file.
3) To add the child form to the parent, we must create a new child form object; set its Mdi-Parent property to the parent form, and call method Show.

Note:

The code to create a child (step 3), usually lies inside an event handler, which creates a new window in response to a user action.

Multiple-Document-Interface (MDI) Windows

To create an MDI form:

1) create a new Form and set its IsMDIContainer property to True
2) create a child form class to be added to the form. (To do this, right-click the project in the Solution Explorer, select Add Windows Form... and name the file.
3) To add the child form to the parent, we must create a new child form object; set its Mdi-Parent property to the parent form, and call method Show.

Note:

The code to create a child (step 3), usually lies inside an event handler, which creates a new window in response to a user action.

Multiple-Document-Interface (MDI) Windows

Common MDI Child Properties

IsMdiChild Indicates whether the Form is an MDI child. If True, Form is an MDI child (read-only property).

MdiParent Specifies the MDI parent Form of the child.

Common MDI Parent Properties

ActiveMdiChild Returns the Form that is the currently active MDI child (returns null if no children are active).

IsMdiContainer Indicates whether a Form can be an MDI. If True, the Form can be an MDI parent. Default is False.

MdiChildren Returns the MDI children as an array of Forms.

Multiple-Document-Interface (MDI) Windows

Common Method

LayoutMdi Determines the display of child forms on an MDI parent. Takes as a parameter an MdiLayout enumeration with possible values ArrangeIcons, Cascade, TileHorizontal and TileVertical.

Controls

Add Control to form Programmatically:

1. Define an object from the control
2. Create an object from the control
3. Set the object properties
4. Add the object to the form

Example: Add a button to a form:

1. Button btn;
2. btn = new Button();
3. btn.Text = “Press the Button”;
   btn.Location = new Point(80, 80);
4. this.Controls.Add(this.btn);

Controls

Add Control to form Using the Windows Forms Designer:

1. From the ToolBox, Drag the Control that you want to add to the form and place it in the Location you need it.
2. The windows Form designer will generate the code.
3. The windows Form designer usually put the code that it generates in a function named InitializeComponent(), and this is the function to be called in the form constructor

Setting the control property:

From Design View:

Select the control, set the property from the property window

Programmatically:

Write the control_name.the property_name = value

Dialog Box

The DialogBox is a kind of form that is used to set some properties.
There is a lot of defined dialogBox “Common Dialog Box” such as “Open, Save, …)
Also, the user can make his own DialogBox.
There are two types of DialogBox:

- ***Modal DialogBox :***

it is the most common, it changes the mode of i/p from the main application to the dialog box. The user can’t switch between DialogBox and the application until he end explicitly the DialogBox (Ok/Cancel).

- ***Modeless DialogBox :***

here the user can switch between the DialogBox and the Application.

Dialog Box

•*Modal DialogBox :*

it is the most common, it changes the mode of i/p from the main application to the dialog box. The user can't switch between DialogBox and the application

Programmers often use Modal Dialog Box when program needs to obtain information from a user beyond what can be easily managed in Menu.

To Create Modal DialogBox:

1. Create class for DialogBox (Has 2 buttons Ok/Cancel)
2. Create class for Application Create object from Dialog Class
3. Call ShowDialog (Invoke the Modal)

There is a property of the dialogbox form named DialogResult of type DialogResult which is enumeration.

Dialog Box

•*Modal DialogBox :*

In either case select Ok/Cancel, the dialog box is closed “disappear from the screen” and return to the point it is called from.

When closing, the ShowDialog() return the value of the DialogResult property

Although the dialogbox has been terminated and no longer visible, the dialogbox object is still valid. This means that we can access all member of the dialogBox.

•*Modeless DialogBox:*

There is a property named owner, set its value to the Application name. Before calling Show() method

Note: Don’t forget to set the value of the DialogResult of the DialogBox to DialogBox.Ok / DialogBox.Cancel in the handel of Ok/Cancel buttons.

Common Dialog Box

C# contains a set of common dialog boxes, which are ready made dialog box (Color, Open, Save, …).

All common dialog boxes are “Modal dialog box”, and are treated as any modal dialog box.

The dealing with common dialog box, is like other dialog box except you are not going to create the dialog box.

I.E.

In the event that will display the dialog box, you have to:

1) Create an object from the dialog box
2) Call the ShowDialog() method
3) Handle the value which return from ShowDialog(), even Ok or Cancel

Tab Control

The TabControl control creates tabbed windows. This allows the programmer to design user interfaces that fit a large number of controls or a large amount of data without using up valuable screen “real estate.”

TabControls contain TabPage objects which can contain controls.

1) add the TabPages to the TabControl.
2) add controls to the TabPage objects

Only one TabPage is displayed at a time. Programmers can add TabControls visually by dragging and dropping them onto a form in design mode.

To add TabPages in the Visual Studio .NET designer, right-click the TabControl, and select Add Tab.

Alternatively, click the TabPages collection in the Properties window, and add tabs in the dialog that appears.

Tab Control

TabControl properties:

ImageList:	Specifies images to be displayed on a tab
ItemSize:	Specifies tab size.
MultiLine:	Indicates whether multiple rows of tabs can be displayed
SelectedIndex:	Indicates index of TabPage that is currently selected
SelectedTab:	Indicates the TabPage that is currently selected
TabCount:	Returns the number of tabs.
TabPages:	Gets the collection of TabPages within our TabControl

User Defined Controls

The .NET Framework allows programmers to create custom controls that inherit from a variety of classes.

These custom controls appear in the user's Toolbox and can be added to Forms, Panels or GroupBoxes in the same way that we add Buttons, Labels, and other predefined controls.

- The simplest way to create a custom control is to derive a class from an existing Windows Forms control, such as a Label.

  This is useful if the programmer wants to include functionality of an existing control, rather than having to reimplement the existing control in addition to including the desired new functionality.

For example, we can create a new type of label that behaves like a normal Label but has a different appearance. We accomplish this by inheriting from class Label and overriding method On-Paint.

User Defined Controls

- To create a new control composed of existing controls, use class UserControl.

  Controls added to a custom control are called constituent controls.

  For example, a programmer could create a UserControl composed of a button, a label and a text box, each associated with some functionality (such as that the button sets the label's text to that contained in the text box).

  The UserControl acts as a container for the controls added to it.

  Method OnPaint cannot be overridden in these custom controls—their appearance can be modified only by handling each constituent control's Paint event.

User Defined Controls

- A programmer can create a brand-new control by inheriting from class Control.

  This class does not define any specific behavior; that task is left to the programmer.

  Instead, class Control handles the items associated with all controls, such as events and sizing handles.

  Method OnPaint should contain a call to the base class's OnPaint method, which calls the Paint event handlers.

  The programmer must then add code for custom graphics inside the overridden OnPaint method.

  This technique allows for the greatest flexibility