COMPSCI 280 S1 2011
Enterprise Software Development

Developing Windows Applications with Visual Studio 2010
Windows Forms Controls and Coding

Agenda & Reading

- Agenda:
  - Windows Forms
  - Working with Controls
  - Button, Label & TextBox
  - Keyboard Events
  - CheckBox, RadioButton, ComboBox & ListBox
  - DateTimePicker & MessageBox
  - TabControl
  - File Common Dialog Boxes & FolderBrowserDialog
  - Menus & Toolbars
  - MDI Child Forms
  - Validation
- Recommended Reading:
  - Getting Started with Windows Forms
    - http://msdn2.microsoft.com/en-us/library/ms229601(VS.80).aspx
  - Microsoft Visual C# 2008 Step by Step
    - Chapter 20-22
- Hands-On Lab:
  - Lecture14Lab

Windows Form

- A form is a visual surface on which you display information to the user.
- A control is a discrete user interface (UI) element that displays data or accepts data input.
  - When a user does something to your form or one of its controls, the action generates an event.
  - Your application reacts to these events by using code, and processes the events when they occur.
- Windows Forms contains a variety of controls that you can add to forms:
  - Button, CheckBox, RadioButton, and TextBox
  - ComboBox and ListBox
  - Horizontal and vertical scrollbars
  - Panel controls (contain other controls), TabControl
  - ToolStrip and MenuStrip (Toolbars and menus that contain text and images, display submenus, and host other controls such as text boxes and combo boxes)
- You can also create your own custom controls using the UserControl class.

Events of Windows Form

- When a Windows Forms application starts, the startup events of the main form are raised:
  - **Load**

    ```
    this.Load += new System.EventHandler(this.Form1_Load);
    ```

    - Occurs before a form is displayed for the first time.
- When an application closes by calling the close(), the shutdown events of the main form are raised:
  - **FormClosing**
    - Occurs as the form is being closed. When a form is closed, it is disposed, releasing all resources associated with the form
    - To cancel the closure of a form, set the Cancel property of the FormClosingEventArgs passed to your event handler to true.

    ```
    private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
       e.Cancel = true; //Cancel closing form
    ```

  - **FormClosed**
    - occurs after the form has been closed by the user or by the Close method or the Exit method of the Application class..

Controls

- To add controls to the Windows Forms
  - Using the Windows Forms Designer
    - Drag and drop the control onto the form
      - Note: For non-visual control (or component), they are displayed at the component tray at the bottom of the form)
  - To add control to the form programmatically
    - Create the control and set the location of the control
    - Add the control to the collection of controls contained within the form, or to a container control, such as Panel, GroupBox…

Avoid performing layout on the form while it is being created and populated with controls

```
SuspendLayout();
Label MyLabel = new Label();
MyLabel.Text = "Sample";
MyLabel.Location = new Point(464, 135);
Panel1.Controls.Add(MyLabel);
ResumeLayout();
```

The collection of the panel control

Working with Controls

- To set the text displayed by a control
  - you can set or return the text by using the **Text** property
  - You can change the font by using the **Font** property.
- To set the image displayed by a control
  - Select the **Image** or **BackgroundImage** property of the control, then click the ellipsis button to display the Select Resource dialog box.
  - Select the image you want to display.
- To set the control can respond to user interaction.
  - Set the **Enabled** property to true
- To set the control is displayed
  - Set the **Visible** property to true
- To create an access key for a control
  - With an access key, the user can "click" a button by pressing the ALT key in combination with the predefined access key
  - Set the Text property to a string that includes an ampersand (**&**) before the letter that will be the access key

Arranging Controls

- To align multiple controls on a form
  - Select the controls and choose Format->Align->Left/Top/…
- To anchor controls on Windows Forms
  - When a control is anchored to a form and the form is resized, the control maintains the distance between the control and the anchor positions
  - Select the controls and set the Anchor property
- To dock controls on Windows Forms
  - You can dock controls to the edges of your form or have them fill the control's container
  - Select the controls and set the Dock property
- To resize controls on Windows Forms
  - To resize a control
    - Click the control to be resized and drag one of the eight sizing handles.
  - To resize multiple controls on a form
    - Select the controls and choose Format-> Make Same Size->Width/height/both
- To set the tab order on Windows Forms
  - To set the order
    - Choose View->Tab Order
    - Click the controls sequentially to establish the tab order you want, or
    - Select the control and set the TabIndex property (index starts from 0)
  - To remove a control from the tab order
    - Set the control's TabStop property to false

Button, Label & TextBox

- Button
  - Allows the user to click it to perform an action
  - Properties: **Text** & **Image**
    - You can set a button on the form that is clicked when the user presses the ENTER key using the AcceptButton property of the form.
  - Event: **Click**: Occurs when the Button control is clicked.
- Label
  - Display text or images that cannot be edited by the user
  - Properties: **Text** & **Image**
- TextBox
  - Gets input from the user or to display text.
  - Properties: **Text** & **Multiline**
    - Insertion Point
      - To make the text box insertion point visible by default
        - Set the TextBox control's TabIndex property to 0.
    - To create a password text box
      - Set the PasswordChar property
  - Event:
    - **TextChanged**: Occurs when the Text property value changes.

Keyboard Events

- There are 3 keyboard related events that can occur on a control. They occur in the following order:
  - A **KeyDown** event.
    - Occurs when a key is pressed while the control has focus.
  - A **KeyPress** event
    - Occurs when a key is pressed while the control has focus.
  - A **KeyUp** event
    - Occurs when a key is released while the control has focus.
- You can use the KeyDown event to determine the type of character entered into the control.
  - A KeyEventArgs parameter, which provides
    - The KeyCode property (which specifies a physical keyboard button)
    - The Modifiers property (SHIFT, CTRL, or ALT)

```
if (e.Modifiers == Keys.Alt && e.KeyCode == Keys.A)
   Console.WriteLine("Press Alt-A");
```

CheckBox

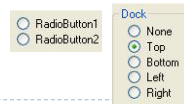
- The Checkbox control indicates whether a particular condition is on/off (True/False)

- Properties:
  - The **Checked** property returns either true or false.
  - The **CheckState** property
    - Returns either Checked or Unchecked; or,
    - Returns Checked or Unchecked or Indeterminate if ThreeState property is set to true. (The Checked property returns true for both Checked and Indeterminate.)
- Events:
  - The **Click** event occurs when the control is clicked.
  - The **CheckedChanged** event occurs when the value of the Checked property changes.

```
private void CheckBox1_CheckedChanged(object sender, EventArgs e) {
   MessageBox.Show(CheckBox1.CheckState.ToString());
}
private void CheckBox1_Click(object sender, EventArgs e) {
  MessageBox.Show(CheckBox1.CheckState.ToString());
}
```

- Note:When the AutoCheck property is true (the default), the CheckBox is automatically selected or cleared when it is clicked. Otherwise, you must manually set the Checked property when the Click event occurs.

RadioButton

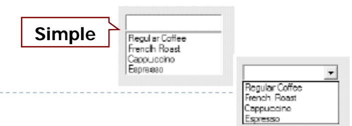


- Windows Forms RadioButton controls present a set of two or more mutually exclusive choices to the user.
- Properties:
  - The **Checked** property returns either true or false.
- Events:
  - The **Click** event occurs when the control is clicked.
  - The **CheckedChanged** event occurs when the value of the Checked property changes.

```
private void RadioButton1_Click(object sender, EventArgs e) {
  RadioButton rdb = (RadioButton)sender;
  MessageBox.Show(rdb.Text);
}
private void rdb_Click(object sender, EventArgs e){
  if (sender == rdbNone)
    btnDemo.Dock = DockStyle.None;
  else if ...
```

- If the AutoCheck property is set to true (the default), when the radio button is selected all others in the group are automatically cleared.

ComboBox



- Display data in a drop-down combo box.
- Properties
  - The **DropDownStyle** property determines the style of combo box to display
    - A simple drop-down, where the list always displays,
    - A dropDownList box, where the text portion is **NOT** editable and you must select an arrow to view the drop-down list box, or
    - The default dropDown list box, where the text portion is editable and the user must press the arrow key to view the list.
  - The **Items** property gets an object representing the collection of the items contained in this ComboBox.
    - The **Items.Count** property reflects the number of items in the list
  - The **SelectedIndex** property gets or sets the index specifying the currently selected item.
  - The **SelectedItem** property gets or sets a reference to the object.
  - The **Text** property specifies the string displayed in the editing field
  - The **SelectedText** property gets or sets the text that is selected in the editable portion of a ComboBox.
- Event:
  - The **SelectedIndexChanged** event occurs when the SelectedIndex property has changed.

ComboBox (con't)

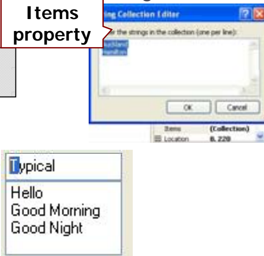
- To add items
  - Using the Add/Insert method
  - Using the AddRange method

```
ComboBox1.Items.Add("Hamilton");
ComboBox1.Items.Insert(0, "Auckland");
```

```
string[] cities = new string[] {"Auckland", "Hamilton"};
ComboBox1.Items.AddRange(cities);
```

  - Alternatively, you can add items to the list by using the Items property at design time.
- To remove an item

Items property



```
ComboBox1.Items.RemoveAt(0);
ComboBox1.Items.Remove(ComboBox1.SelectedItem);
```

- To remove all items

```
ComboBox1.Items.Clear();
```

- SelectedText VS Text
  - SelectedText =>Selected text in the editable portion
  - Text => string displayed in the editable portion

Editable portion

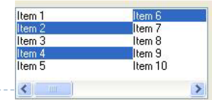
Typical
Hello
Good Morning
Good Night

```
private void ComboBox1_MouseDown(object sender, MouseEventArgs e) {
  if (e.Button == MouseButtons.Right) {
    Console.WriteLine("Text" + ComboBox1.Text);
    Console.WriteLine("SelectedText" + ComboBox1.SelectedText);
  ...
```

Print "Typical"

Print "T"

ListBox



- Displays a list of items from which the user can select one or more.
- Properties:
  - Items, Items.Count, SelectedIndex, SelectedItem
  - The **MultiColumn** property gets or sets a value indicating whether the ListBox supports multiple columns.
  - The **SelectionMode** property determines how many list items can be selected at a time.
    - SelectionMode.**One**: Only one item can be selected.
    - SelectionMode.**MultiExtended**: Extending the selection by pressing SHIFT/CTRL
    - SelectionMode.**MultiSimple**: A mouse click or pressing the SPACEBAR selects or deselects an item in the list.
  - The **SelectedIndices** property gets a collection that contains the zero-based indexes of all currently selected items in the ListBox.
  - The **SelectedItems** property gets a collection containing the currently selected items in the ListBox.

```
foreach (Object i in ListBox1.SelectedItems)
  Console.Write(i.ToString() + " ");
```

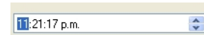
```
Item 2 Item 4 Item 6
```

DateTimePicker & MessageBox

- The DateTimePicker control allows the user to select a single item from a list of dates or times.
  - The **Value** property contains the current date and time the control is set to.
    - Four formats: which are set by the Format property:
      - Long (Monday, 27 February 2006), Short (27/02/2006), Time(10:15:48 p.m.), or Custom

```
MessageBox.Show("The selected value is " + DateTimePicker1.Text);
```

  - To display Time with the DateTimePicker Control
    - Set the Format property to Time
    - Set the ShowUpDown property for the DateTimePicker to true.



- The MessageBox control displays a message box that can contain text, buttons, and symbols that inform and instruct the user.
  - **MesageBoxButtons**: defines the number & caption of the buttons appearing
    - OK, OKCancel, YesNo, YesNoCancel …
  - **MessageBoxIcon**: defines the icon appear in message box
    - Error, Warning, Question, Information …
  - **DialogResult**
    - Cancel, OK, Yes …

```
DialogResult returnValue = MessageBox.Show(message, title, buttons, icon);
```

Adding Windows Forms

- To add a new Windows Form
  - Choose Project->Add Windows Forms
- To change the Startup form in Windows Forms
  - Choose Project->Properties, select the form from the Startup form drop-down list
- To display the form in modal or modeless.
  - Forms and dialog boxes are either modal or modeless.
    - A **modal** form or dialog box must be closed or hidden before you can continue working with the rest of the application.
      - Call the ShowDialog method.

Form.ShowDialog()

```
Form2 f2 = new Form2();
f2.ShowDialog();
```

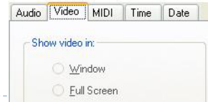
    - **Modeless** forms let you shift the focus between the form and another form without having to close the initial form.
      - Call the Show method.

```
Form2 f2 = new Form2();
f2.Show();
```

Form.Show()

- Note: MessageBox is modal
  - You must close the message box before another form in the program can get input focus.
  - You cannot create a new instance of the MessageBox class. To display a message box, call the static method **MessageBox.Show**

The TabControl Control



- The Windows Forms TabControl displays multiple tabs, like dividers in a notebook or labels in a set of folders in a filing cabinet.
  - The tabs can contain pictures and other controls.
  - The most important property of the TabControl is TabPages, which contains the individual tabs.
- Properties:
  - The **SelectedIndex** property gets or sets the index of the currently selected tab page.
  - The **SelectedTab** property gets or sets the currently selected tab page.
  - The **TabCount** property gets the number of tabs in the tab strip.
  - The **TabPages** property gets the collection of tab pages in this tab control.
- Methods
  - The **SelectTab** method makes the specified tab the current tab.
  - The **DeselectTab** method makes the tab following the tab with the specified index the current tab.
- Events
  - The **Deselecting** event occurs before a tab is deselected, enabling a handler to cancel the tab change.
  - The **Deselected** event occurs when a tab is deselected.
  - The **Selecting** event occurs before a tab is selected, enabling a handler to cancel the tab change.
  - The **Selected** event occurs when a tab is selected.

The TabControl Control (con't)

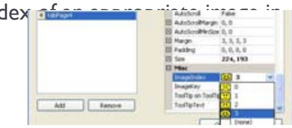
- To add a tab programmatically

```
TabPage myTabPage = new TabPage("NewPage"));
TabControl1.TabPages.Add(myTabPage);
```

  - Alternatively, you can add tabs to the TabControl by using the **TabPages** property at design time
- To remove a tab programmatically

```
TabControl1.TabPages.Remove(TabControl1.SelectedTab);
```

- To display an icon on the label part of a tab
  - Add an **ImageList** control and add images to the image list.
  - Set the ImageList property of the TabControl to the ImageList control.
  - Set the ImageIndex property of the TabPage to the index of an appropriate image in the list.



```
TabPage1.Controls.Add(new Button());
```

- To add a Control to a Tab Page
  - To add a control programmatically
    - Use the Add method of the collection returned by the **Controls** property of a TabPage:
    - Alternatively, you can add controls to the TabControl at design time

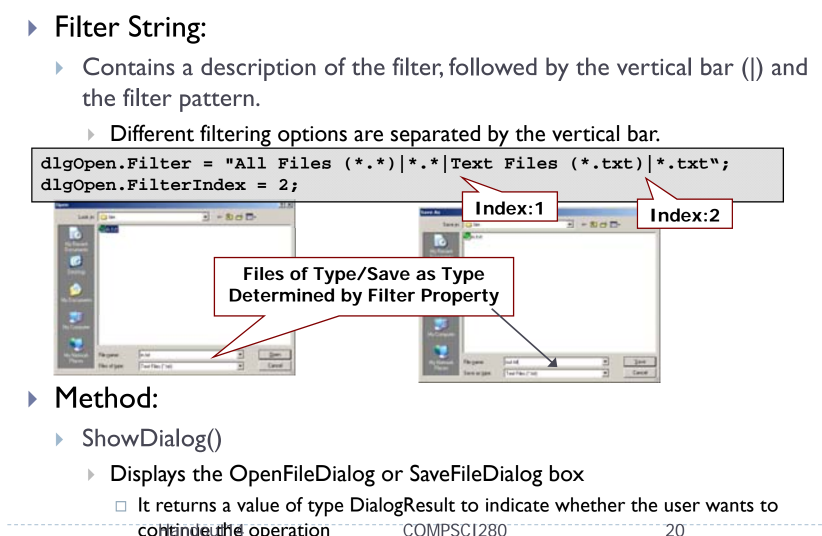
File Common Dialog Boxes

- The previous data file examples all included "hard-coded" file names and paths
- It may be preferable to allow the user to locate the file to open at run time
  - **OpenFileDialog** Common Dialog
    - Prompts the user to open a file.
  - **SaveFileDialog** Common Dialog
    - Prompts the user to select a location for saving a file.
- Properties
  - FileName
    - Gets/sets a string containing the file name selected in the file dialog box.
  - Filter
    - Gets/sets the current file name filter string, which determines the choices that appear in the "Save as file type" or "Files of type" box in the dialog box
  - FilterIndex
    - Gets/sets the index of the filter currently selected in the file dialog box.

File Dialog Boxes

- Filter String:
  - Contains a description of the filter, followed by the vertical bar (|) and the filter pattern.
    - Different filtering options are separated by the vertical bar.

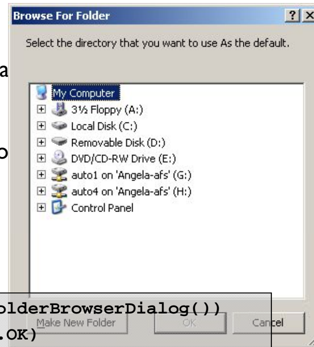
```
dlgOpen.Filter = "All Files (*.*)|*.*|Text Files (*.txt)|*.txt";
dlgOpen.FilterIndex = 2;
```



- Method:
  - ShowDialog()
    - Displays the OpenFileDialog or SaveFileDialog box
      - It returns a value of type DialogResult to indicate whether the user wants to continue the operation

The FolderBrowserDialog Control

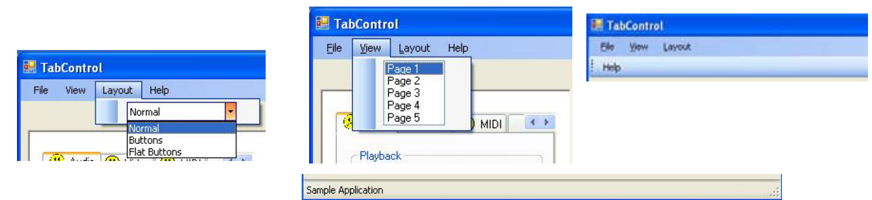
- Represents a common dialog box that allows the user to choose a folder
- Properties:
  - Description
    - Gets or sets the descriptive text displayed above the tree view control in the dialog box.
  - RootFolder
    - Gets or sets the root folder where the browsing starts from.
  - SelectedPath
    - Gets or sets the path selected by the user.




```
using (FolderBrowserDialog dlgFolder = new FolderBrowserDialog())
  if (dlgFolder.ShowDialog() == DialogResult.OK)
    txtDirectory.Text = dlgFolder.SelectedPath;
}
```

Menus & Toolbars

- ToolStrip controls are toolbars that can host menus, controls, and user controls in your Windows Forms applications.
- The **ToolStrip** and its associated classes enable you to create toolbars and other user interface elements that can have a Microsoft® Windows® XP, Microsoft Office, Microsoft Internet Explorer, or custom appearance and behavior.
- The **MenuStrip** control allow you to create easily customized, commonly employed menus that support advanced user interface and layout features
- The **StatusStrip** control is used on forms as an area, usually displayed at the bottom of a window, in which an application can display various kinds of status information.



ToolStripItem

- The ToolStripItem represents the base class that manages events and layout for all the elements that a ToolStrip or ToolStripDropDown can contain.
  - **ToolStripLabel**: Used to display normal text, hyperlinks, and images.
  - **ToolStripButton**: Provides a typical pushbutton that you can configure to support both text and images.
  - **ToolStripComboBox**: A ComboBox with methods/properties to configure various styles.
  - **ToolStripSeparator**: A separator that you can use to visually separate groups of ToolStripItem elements.
  - **ToolStripDropDownButton**: This control provides a button which, when clicked, displays a ToolStripDropDown control.
  - **ToolStripTextBox**: A normal textbox which can be used to enter text.
  - **ToolStripMenuItem**: A special menu control built specifically for use with the MenuStrip and ContextMenuStrip controls.
  - **ToolStripProgressBar**: A specialized progress bar implementation for use within a StatusStrip control.
  - **ToolStripSplitButton**: A combination of normal button and a DropDownButton.
  - **ToolStripControlHost**: A control that acts as a host to your customized implementations or any other Windows Form controls.

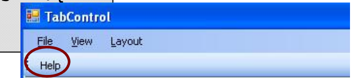
The ToolStrip Control

ToolStripButton
ToolStripSeparator
ToolStripLabel
ToolStripDropDownButton
ToolStripSplitButton
ToolStripTextBox
ToolStripComboBox

- To Add an item programmatically

```
mnuHelp = new ToolStripMenuItem("Help");
mnuStrip.Items.Add(mnuHelp);
mnuHelp.Click += mnuHelp_Click;
```

```
private void mnuHelp_Click(object sender, EventArgs e){
  MessageBox.Show("HELP");
}
```



- Alternatively, you can add items to the ToolStrip by using the Items property at design time, or click the drop-down list on the toolbar

- To change the layout of the item

```
tsButton1.TextAlign = ContentAlignment.MiddleRight;
tsButton1.ImageAlign = ContentAlignment.MiddleRight;
tsButton1.TextImageRelation = TextImageRelation.ImageAboveText;
```

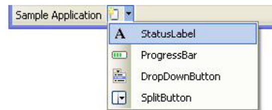
The StatusStrip Control

ToolStripStatusLabel
ToolStripDropDownButton
ToolStripSplitButton
ToolStripProgressBar

- To Add an item programmatically
  - StatusLabel & progressBar

```
tslabel = new ToolStripStatusLabel("Sample Only");
StatusStrip1.Items.Add(tslabel);
tsProgressBar = new ToolStripProgressBar();
StatusStrip1.Items.Add(tsProgressBar);
```

  - Alternatively, you can add items to the StatusStrip by using the Items property at design time, or click the drop-down list on the StatusStrip

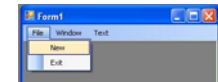


- To adjust the position of the progressBar
  - Set the value property of the prograssBar (0 to 100)

The MenuStrip Control

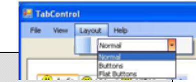
ToolStripMenuItem
ToolStripTextBox
ToolStripComboBox

- To Add an item in the main menu programmatically
  - Use the Items Collection



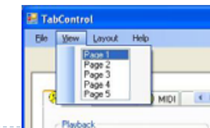
```
mnuFile = new ToolStripMenuItem("File");
mnuStrip.Items.Add(mnuFile);
```

  - Alternatively, you can add items to the MenuStrip by using the Items property at design time, or click the drop-down list on the MenuStrip
- To Add an DropDownMenu programmatically
  - Use the DropDownItems collection of the menu item, or



```
tsAppearanceComboBox = new ToolStripComboBox();
tsAppearanceComboBox.DropDownStyle = ComboBoxStyle.DropDownList;
tsAppearanceComboBox.Items.AddRange(new Object[] {"Normal", "Buttons"});
tsAppearanceComboBox.ComboBox.SelectedIndex = 0;
mnuLayout.DropDownItems.Add(tsAppearanceComboBox);
```

  - Add a control to the ToolStripControlHost



```
lstTabPages = new ListBox();
for (int i = 1; i<=5; i++)
  lstTabPages.Items.Add("Page " + i);
host = new ToolStripControlHost(lstTabPages);
mnuView.DropDownItems.Add(host);
```

MDI applications

- Multiple-document interface (MDI) applications enable you to display multiple documents at the same time, with each document displayed in its own window. MDI applications often have a Window menu item with submenus for switching between windows or documents.
- To Create an MDI parent Form
  - Create a Windows Form
  - Set the IsMDIContainer property to true.
    - This designates the form as an MDI container for child windows.
- To Create MDI Child Forms
  - Add a New Window Form and rename it to ChildForm
  - Add controls to the child form
  - Create a Click event handler on the parent form to create child form
    - Create an instance of the child form
    - Set the MdiParent property of the child form
    - Display the child form

```
ChildForm f = new ChildForm();
f.MdiParent = this;
f.Show();
```

Example:

- Tasks:
  - How to : Create a MDI Parent form with a MenuStrip control
    - Set the **IsMDIContainer** property to true
    - Set the **MidWindowListItem** property to a ToolStripMenuItem
      - It is used to display a list of Multiple-document interface (MDI) child forms
  - How to: Create MDI Child Forms
    - Create a child form template
    - Implement the event handler to create a child form
      - Set the **MdiParent** property
  - How to: Determine the Active MDI Child
    - Use the **ActiveMdiChild** property to determine the active child
  - How to: Send Data to the Active MDI Child
    - Use the **activeChild.ActiveControl** to get the active control
  - How to: Arrange MDI Child Forms
    - Use the **LayoutMdi** method to arrange the multiple-document interface (MDI) child forms within the MDI parent form
      - Cascade, TileHorizontal, TileVertical …

ErrorProvider

- ErrorProvider presents a simple mechanism for indicating to the end user that a control on a form has an error associated with it.
  - If an error description string is specified for the control, an icon appears next to the control. The icon flashes in the manner specified by BlinkStyle, at the rate specified by BlinkRate.
  - When the mouse hovers over the icon, a ToolTip appears showing the error description string.
- To Create a ErrorProvider control
  - Drag and drop the control onto the form
    - The ErrorProvider is displayed at the component tray at the bottom of the form
- To set the error message

```
ErrorProvider1.SetError(TextBox1, "Not a number!");
```

- To clear the error message

```
ErrorProvider1.SetError(TextBox1, "");
```

Validation

- When a user enters or leaves a field by using the keyboard, the following events occur in order:
  - Enter (the focus is about to enter the control)
  - GotFocus (the focus has entered the control)
  - Leave (the focus is about to leave the control)
  - Validating (the data in the control is ready to validate)
  - Validated (the data has been validated)
  - LostFocus (the focus has left the control)
  - Note: if you change the focus by using the mouse, the events occur in the following order:
    - Enter->GotFocus->LostFocus->Leave->Validating->Validated.
- You can use the Validating event for validating user input.
  - If validation fails, you can halt the chain of events and prevent the user from moving on until any error is corrected.

```
private void TextBox1_Validating(object sender, CancelEventArgs e) {
...
```

  - Control.CausesValidaton
    - If the CausesValidation property of the control is set to false, the Validating and Validated events are suppressed

Validating with ErrorProvider

- To validate input from the user
  - Create an event handler for the Validating event
  - Validate the input from the user
    - If validation fails, display an ErrorProvider, and the event is canceled by setting the Cancel property of the CancelEventArgs to true. All events that would usually occur after the Validating event are suppressed.

```
string strEntered = TextBox1.Text;
if (! String.IsNullOrEmpty(strEntered)) {
  int number = int.Parse(strEntered);
  if (number < 0) {
    e.Cancel = true;
    errorProvider1.SetError(TextBox1, "less than zero");
  }
  else{
     errorProvider1.Clear();
  }
}
```

Note:

- Closing a form fires the Validating event. If the Cancel property is True in a Validating event, this will prevent the form from closing.
- You can change it in the Closing event of the form by setting the Cancel property to false