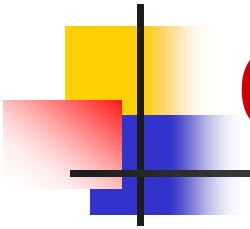
Chapter 5. Advanced GUIs

A reference of MSDN Library for Visual Studio 2017

IT Faculty, TDM University

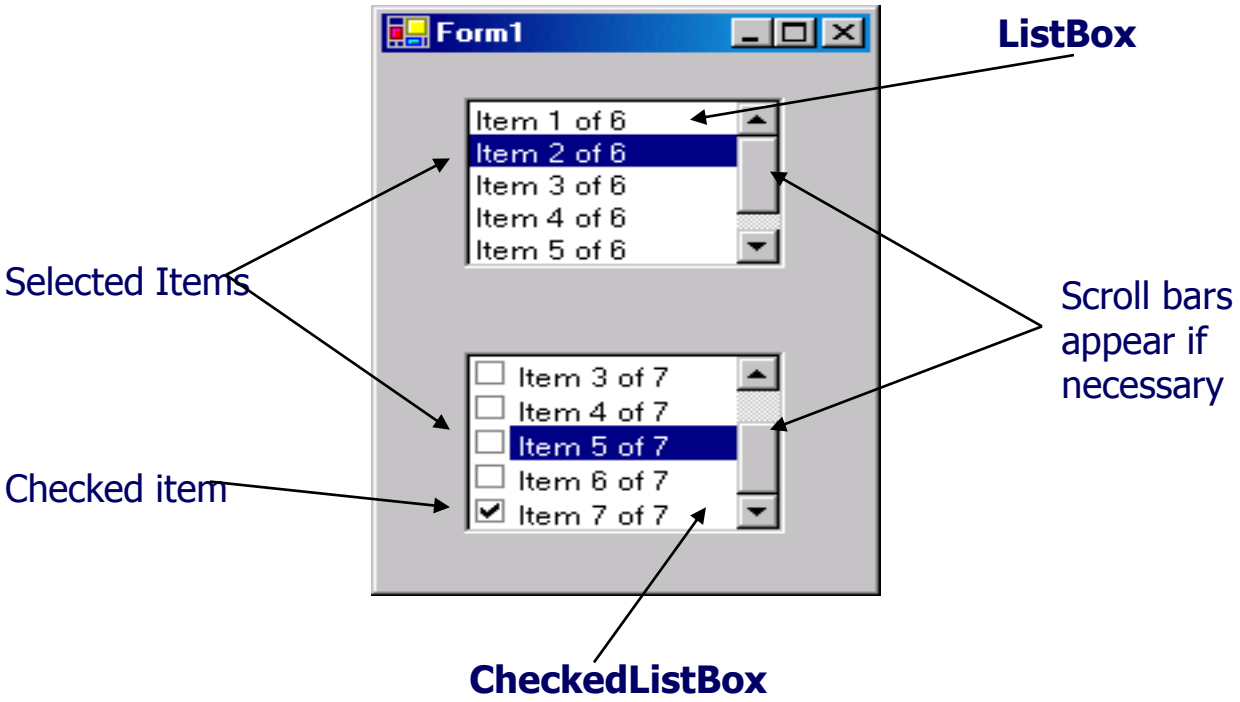


Contents

- ListBox and CheckedListBox
- ComboBox
- TabControl
- Menu and Toolbar
- MDI Windows
- TreeView
- ListView
- LinkLabel

ListBox and CheckedListBox

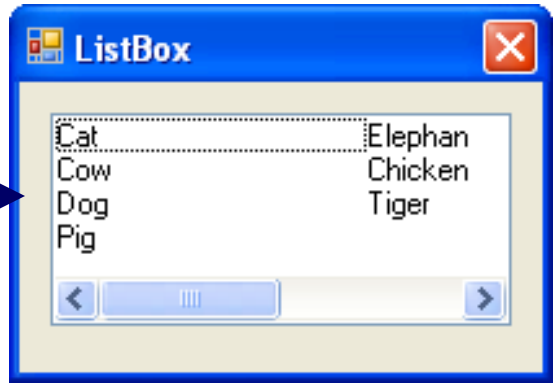
- **`ListBoxes`**
  - Allow users to view and select from items on a list



ListBox and CheckedListBox

- **`ListBox`** Common Properties
  - **`Items`**: Lists the collection of items within the **`ListBox`**.
  - **`MultiColumn`**: Indicates whether the **`ListBox`** can break a list into multiple columns.

```
MultiColum = true
```



ListBox and CheckedListBox

- **`ListBox`** Common Properties
  - **`SelectedIndex`**: Returns the index of the currently selected item. If the user selects multiple items, this method arbitrarily returns one of the selected indices; if no items have been selected, the method returns **`-1`**.
  - **`SelectedIndices`**: Returns a collection of the indices of all currently selected items.

ListBox and CheckedListBox

- **`ListBox`** Common Properties
  - **`SelectedItem`**: Returns the currently selected item.
  - **`SelectedItems`**: Returns a collection of the currently selected item(s).
  - **`SelectedValue`**: Returns the value of the member property specified by the **ValueMember** property.

ListBox and CheckedListBox

- **`ListBox`** Common Properties
  - **`Sorted`**: Indicates whether items appear in alphabetical order. **`True`** causes alphabetization; default is **`False`**.
  - **`SelectionMode`**: Determines the number of items that can be selected and the means through which multiple items can be selected. Values **`None`**, **`One`**, **`MultiSimple`** (multiple selection allowed) and **`MultiExtended`** (multiple selection allowed via a combination of arrow keys, mouse clicks and *Shift* and *Control* buttons).

ListBox and CheckedListBox

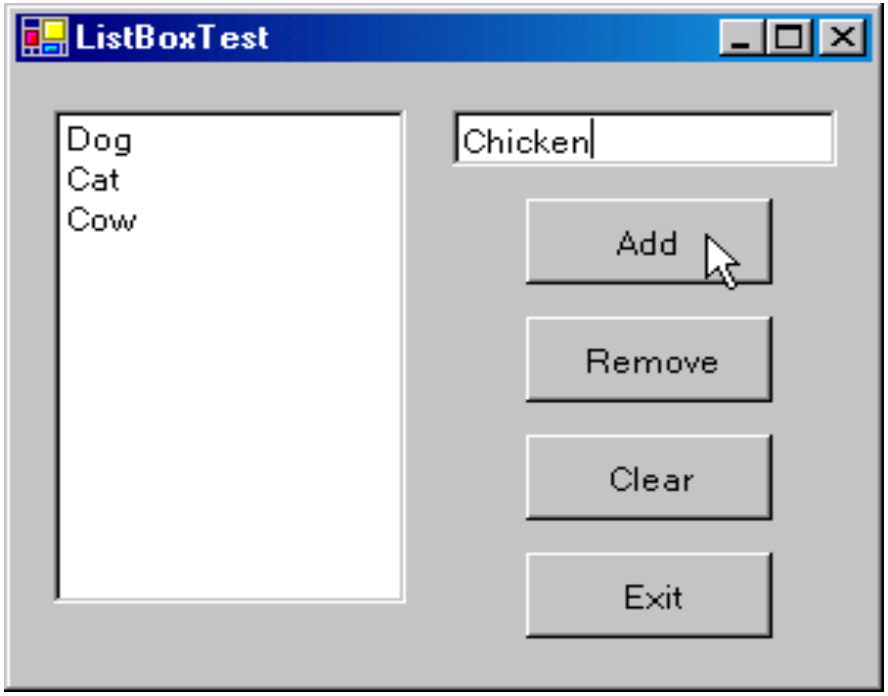
- **`ListBox`** Common Methods
  - **`GetSelected`**: Takes an index, and returns **`True`** if the corresponding item is selected.
  - **`Add`**: Adds an **Item** to the list of Items
    - listBox1.**`Items.Add`**("One");
    - listtBox1.**`Items.Add`**("Two");
  - **`RemoveAt`**: Removes the **Item** at the specified index within the collection
    - listBox1.**`Items.RemoveAt`**(row);
  - **`Clear`**: Clear to all **Items** collection
    - listBox1.**`Items.Clear();`**

ListBox and CheckedListBox

- **`ListBox`** Common Events
  - **`SelectedIndexChanged`**: Occurs when selected index changes. Default when control is double clicked in designer.

ListBox and CheckedListBox

- **`ListBox`** Example



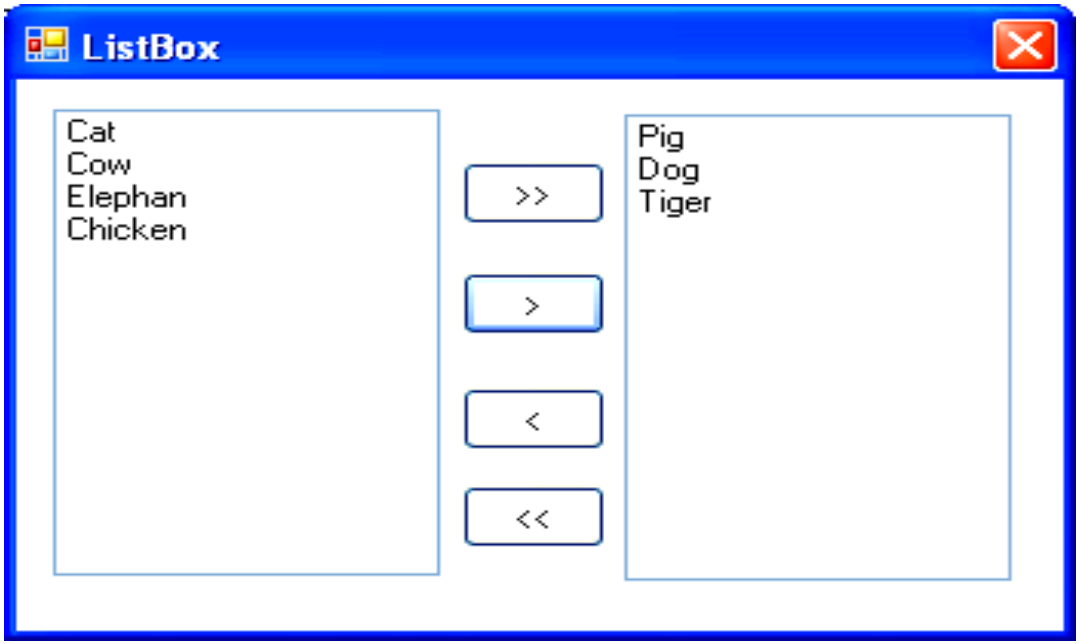
ListBox and CheckedListBox

- Code Pattern

```
private void btnAdd_Click(object sender, EventArgs e)
    {
            listBox1.Items.Add(txtInput.Text);
            txtInput.Clear();
    }
 private void btnRemove_Click(object sender, EventArgs e)
    {
            int row=listBox1.SelectedIndex;
            if (row != -1)  listBox1.Items.RemoveAt(row);
    }
private void btnClear_Click(object sender, EventArgs e)
    {
            listBox1.Items.Clear();
    }
```

ListBox and CheckedListBox

- **`ListBox`** Homework
  - Adds and Removes Items



ListBox and CheckedListBox

- **`CheckedListBoxes`**
  - Extends **`ListBox`** by placing a check box at the left of each item.
  - Can select more than one object at one time
  - Can add to, remove from or clear list
  - Can select multiple items from the list

ListBox and CheckedListBox

- **`CheckedListBox`** Common Properties
  - **`CheckedItems`**: The collection of items that are checked. Not the same as the selected items, which are highlighted (but not necessarily checked).
  - **`CheckedIndices`**: Returns indices for the items that are checked.
  - **`SelectionMode`**: Can only have values **`One`** (allows multiple selection) or **`None`** (does not allow multiple selection).

ListBox and CheckedListBox

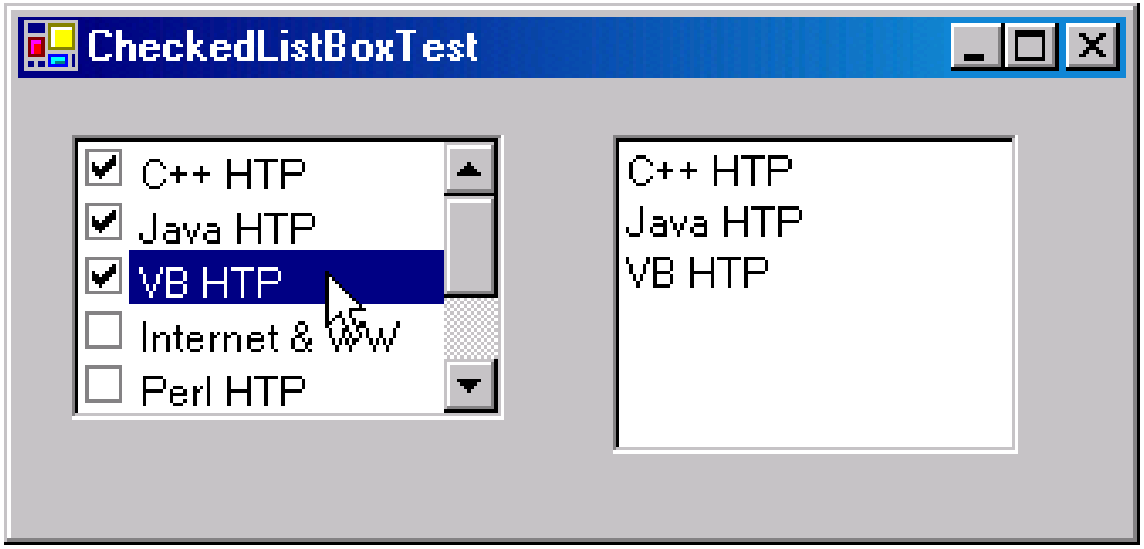
- **`CheckedListBox`** Common Methods
  - **`GetItemChecked`**: Takes an index and returns **`true`** if corresponding item checked.
- **`CheckedListBox`** Common Events
  - **`ItemCheck`**: Occurs when an item is checked or unchecked.

ListBox and CheckedListBox

- **`ItemCheckEventArgs` Properties**
  - **`CurrentValue`**: Whether current item is checked or unchecked. Values **`Checked`**, **`Unchecked`** or **`Indeterminate`**.
  - **`Index`**: Index of item that changed.
  - **`NewValue`**: New state of item.

ListBox and CheckedListBox

- **`CheckedListBox` Example**



ListBox and CheckedListBox

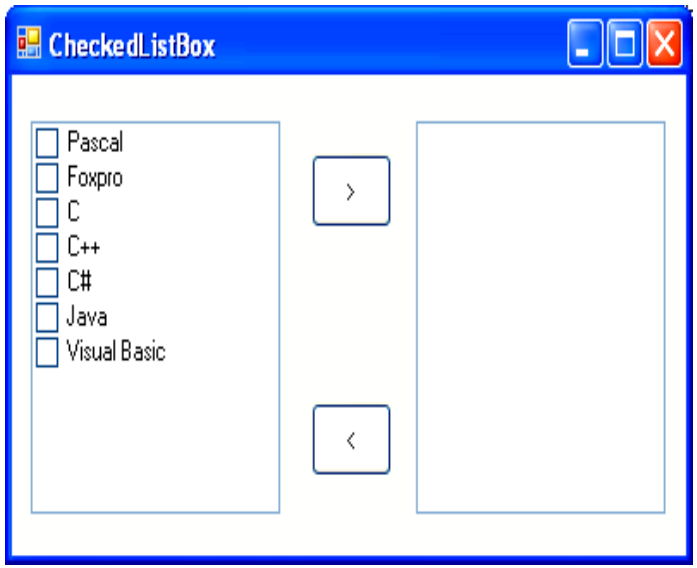
- Code Pattern

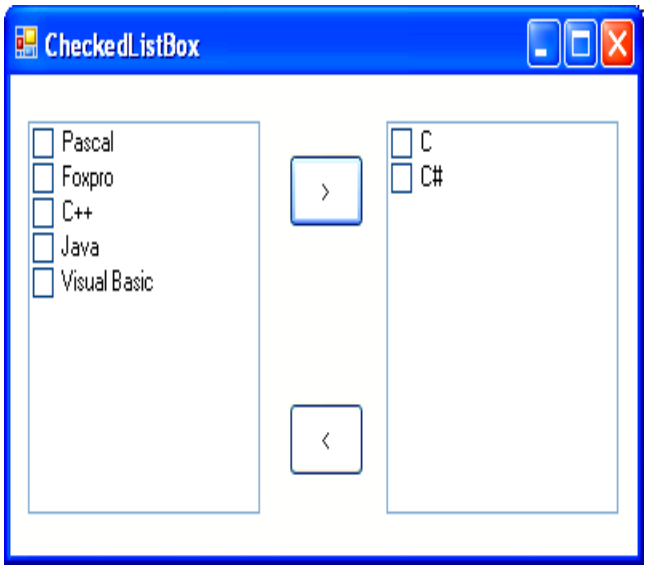
```
private void myCheckedListBox_Load(object sender, EventArgs e)
    {
        checkedListBox1.Items.Add("C++ HTP");
    }
```

```
private void checkedListBox1_ItemCheck(object sender,
                                       ItemCheckEventArgs e)
    {
        string item = checkedListBox1.SelectedItem.ToString();
        if ( e.NewValue == CheckState.Checked )
            listBox1.Items.Add(item);
        else
        listBox1.Items.Remove(item);
    }
```

ListBox and CheckedListBox

- **`CheckedListBox`** Homework





ListBox and CheckedListBox

- Code pattern

```
private void btnAdd_Click(object sender, EventArgs e)
    {
     for (int i = 0; i < checkedListBox1.Items.Count - 1; i++)
         {
            if (checkedListBox1.GetItemChecked(i) == true)
               {
                  // Adds Item i to checkedListBox2
                  string item = checkedListBox1.Items[i].ToString();
                  checkedListBox2.Items.Add(item);
                  // Removes Item i at  checkedListBox1
                  checkedListBox1.Items.RemoveAt(i);
               }
         }
     }
```

ComboBoxes

- Combine **`TextBox`** and drop-down list
- Common Properties
  - **`DropDownStyle`**: Determines the type of combo box. Value **`Simple`** means that the text portion is editable and the list portion is always visible. Value **`DropDown`** (the default) means that the text portion is editable but an arrow button must be clicked to see the list portion. Value **`DropDownList`** means that the text portion is not editable.

ComboBoxes

- Common Properties
  - **`Items`**: Collection of items in the **`ComboBox`** control.
  - **`MaxDropDownItems`**: Maximum number of items to display in the drop-down list (between `1` and `100`). If value is exceeded, a scroll bar appears.
  - **`SelectedIndex`**: Returns index of currently selected item. If there is no currently selected item, `-1` is returned.

ComboBoxes

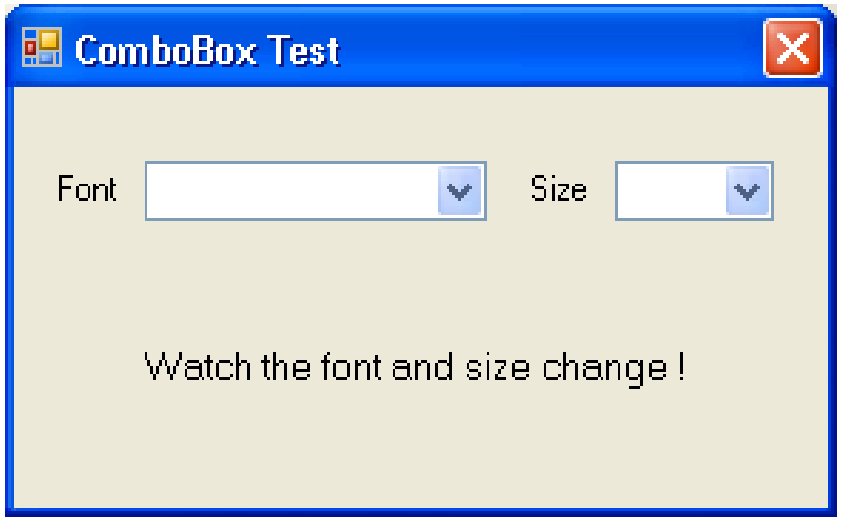
- Common Properties
  - **`SelectedItem`**: Returns the currently selected item.
  - **`Sorted`**: If **`true`**, items appear in alphabetical order. Default **`false`**.
- Common Events
  - **`SelectedIndexChanged`**: Occurs when selected index changes. Default when control double clicked in designer.

ComboBoxes

- **`ComboBox`** Common Methods
  - **`Add`**: Adds an **Item** to the list of Items
    - comboBox1.**`Items.Add`**("One");
    - comboBox1.**`Items.Add`**("Two");
  - **`RemoveAt`**: Removes the **Item** at the specified index within the collection
    - comboBox1.**`Items.RemoveAt`**(row);
  - **`Clear`**: Clear to all **Items** collection
    - comboBox1.**`Items.Clear();`**

ComboBoxes

- Example
  - Design the form



  - **`Note:`** cboFont gets system font

ComboBoxes

- Code Pattern

```
private void ComboBox_Load(object sender, EventArgs e)
        {
                //Get fonts of system
                FontFamily[] ff = FontFamily.Families;
                //Add to cboFont
                for (int i = 0; i <ff.Length; i++)
                    cboFont.Items.Add(ff[i].Name);
                //Add size to cboSize
                for (int i = 8; i <= 72; i++)
                  cboSize.Items.Add(i);
        }
```

ComboBoxes

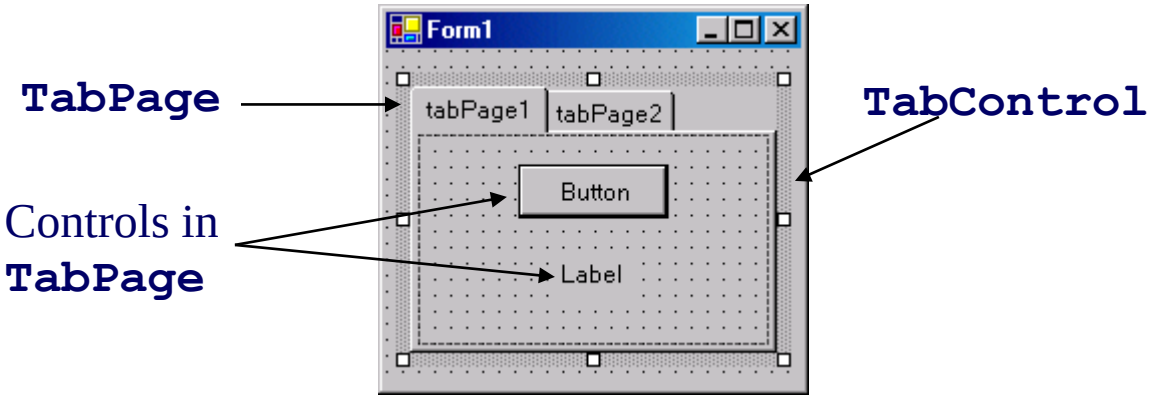
- Code Pattern

```
private void cboFont_SelectedIndexChanged(object sender,
                                          EventArgs e)
    {
     if (cboFont.Text.Trim() == "") return;
         lbDisplay.Font = new Font(cboFont.Text,
                                   lbDisplay.Font.Size);
    }
```

```
private void cboSize_SelectedIndexChanged(object sender,
                                          EventArgs e)
    {
     if (cboSize.Text.Trim() == "") return;
          lbDisplay.Font = new Font(lbDisplay.Font.Name,
                                    (float)Convert.ToDouble(cboSize.Text));
    }
```

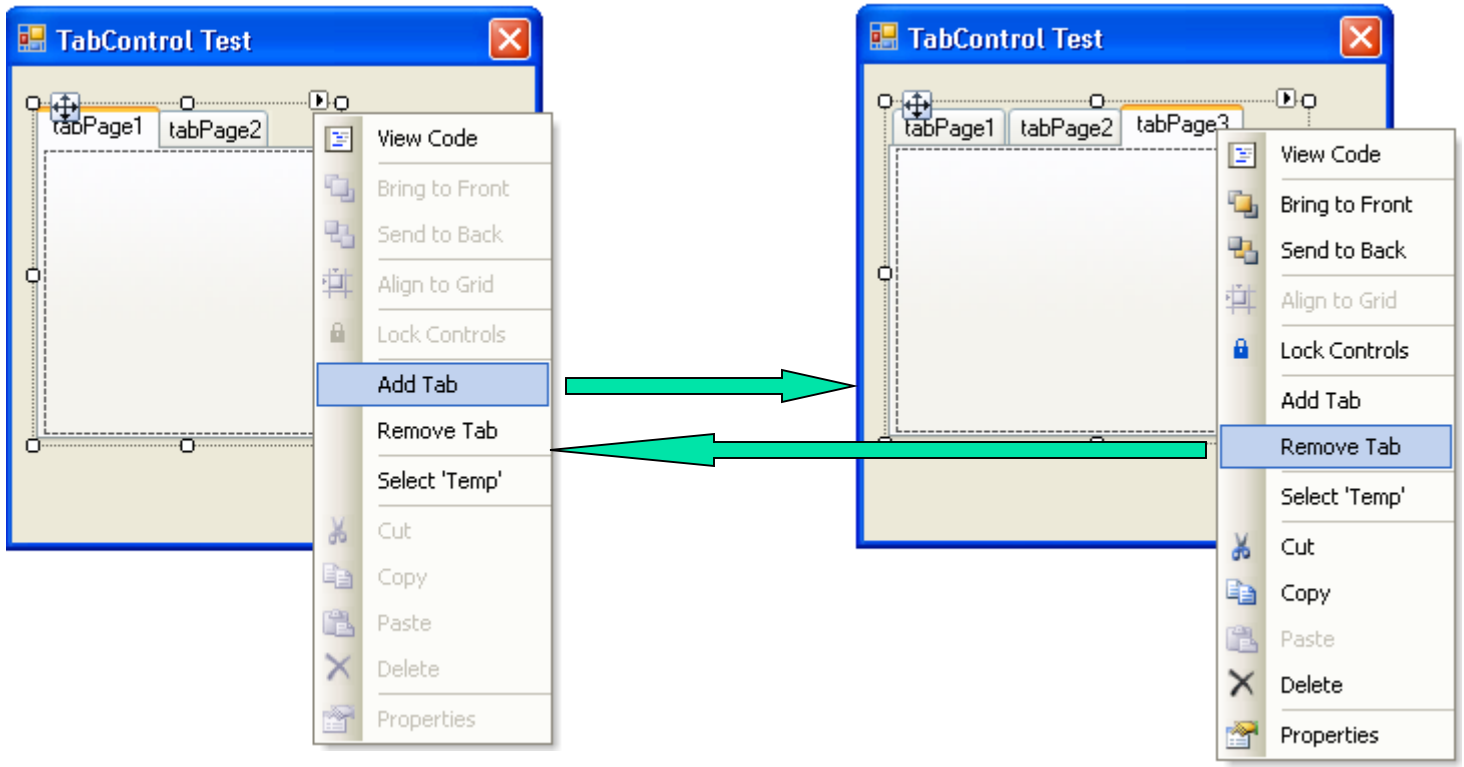
TabControl

- Creates tabbed windows
- Windows called **`TabPage`** objects
  - **`TabPages`** can have controls
  - **`Tabpages`** have own **`Click`** event for when tab is clicked



TabControl

- Add and Remove Tab

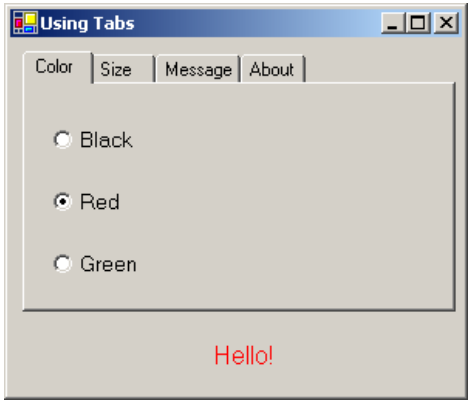


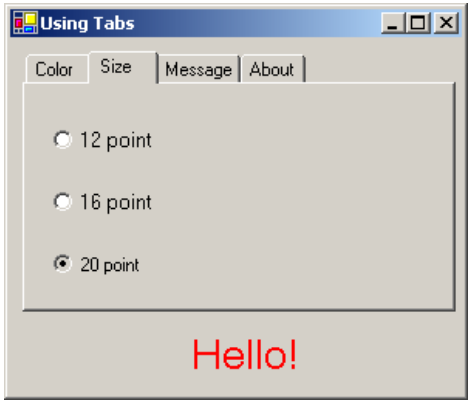
TabControl

- Common Properties
  - **`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`**.
- Common Event
  - **`SelectedIndexChanged`**: Occurs when another **`TabPage`** is selected.

TabControl

- Exercise





TabControl

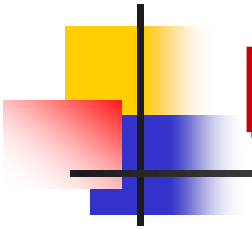
- Code Pattern

```
private void btnBlack_CheckedChanged(object sender, EventArgs e)
    {
        lbDisplay.ForeColor = Color.Black;
    }

private void btnRed_CheckedChanged(object sender, EventArgs e)
    {
        lbDisplay.ForeColor = Color.Red;
    }

private void btnGreen_CheckedChanged(object sender, EventArgs e)
    {
        lbDisplay.ForeColor = Color.Green;
    }
```

TabControl

- Code Pattern

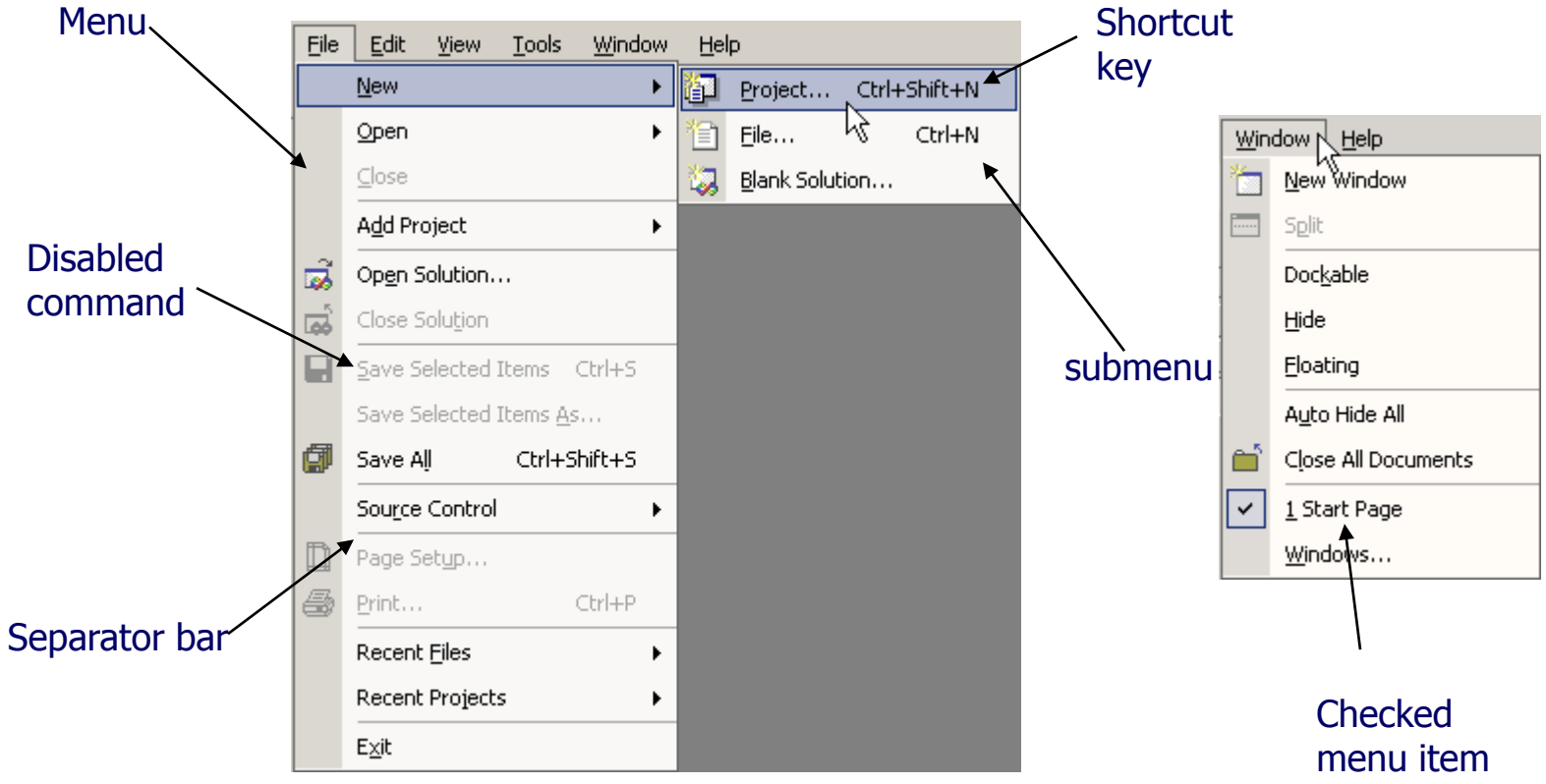
```
private void btnPoint16_CheckedChanged(object sender, EventArgs e)
        {
            lbDisplay.Font = new Font(lbDisplay.Font.Name, 16);
        }

private void btnPoint20_CheckedChanged(object sender, EventArgs e)
        {
            lbDisplay.Font = new Font(lbDisplay.Font.Name, 20);
        }

private void btnHello_CheckedChanged(object sender, EventArgs e)
        {
            lbDisplay.Text = btnHello.Text;
        }
```



Menus

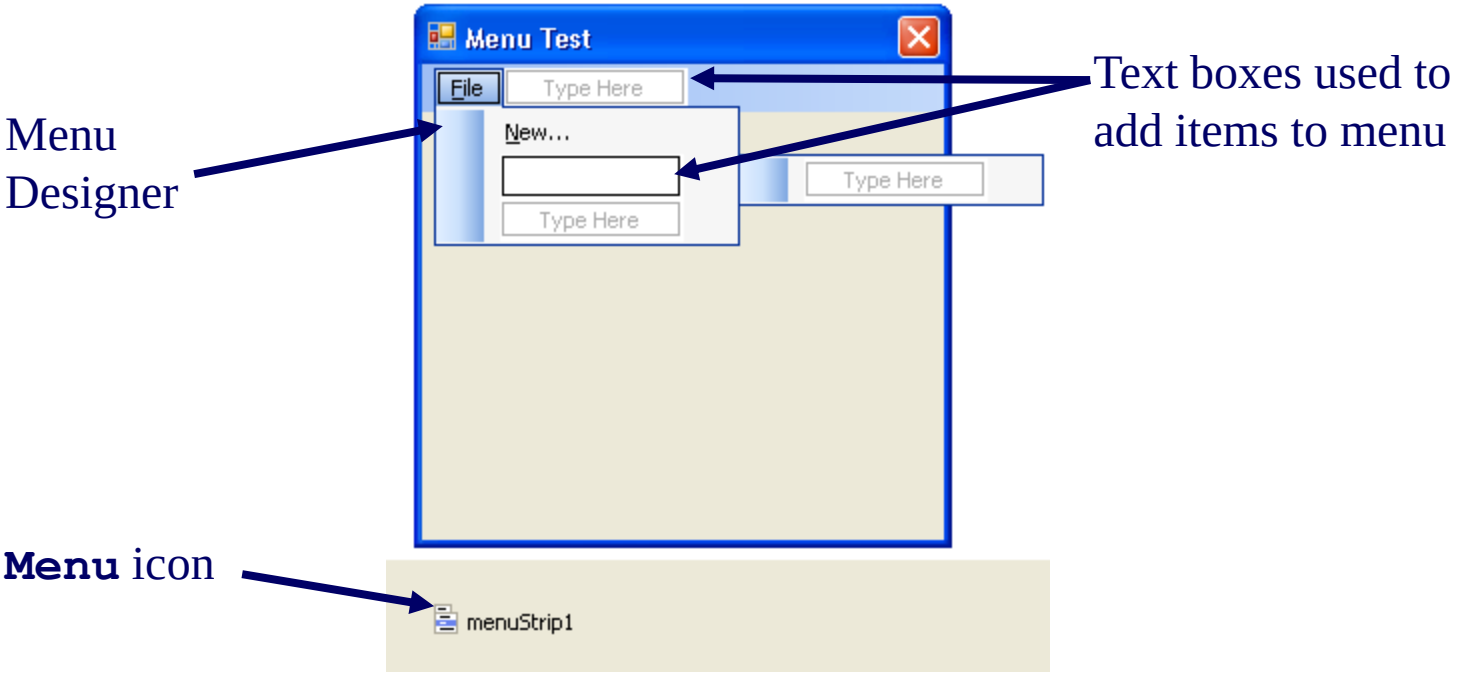
- Group related commands together
- Contain
  - Commands
  - Submenus

Menus


Menu
Shortcut key
Disabled command
submenu
Separator bar
Checked menu item
File
Edit
View
Tools
Window
Help
New
Open
Close
Add Project
Open Solution...
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Source Control
Page Setup...
Print... Ctrl+P
Recent Files
Recent Projects
Exit
Project... Ctrl+Shift+N
File... Ctrl+N
Blank Solution...
Window
Help
New Window
Split
Dockable
Hide
Floating
Auto Hide All
Close All Documents
1 Start Page
Windows...

MenuStrip

- A standard menu of the form is provided with the **`MenuStrip`** control.



MenuStrip

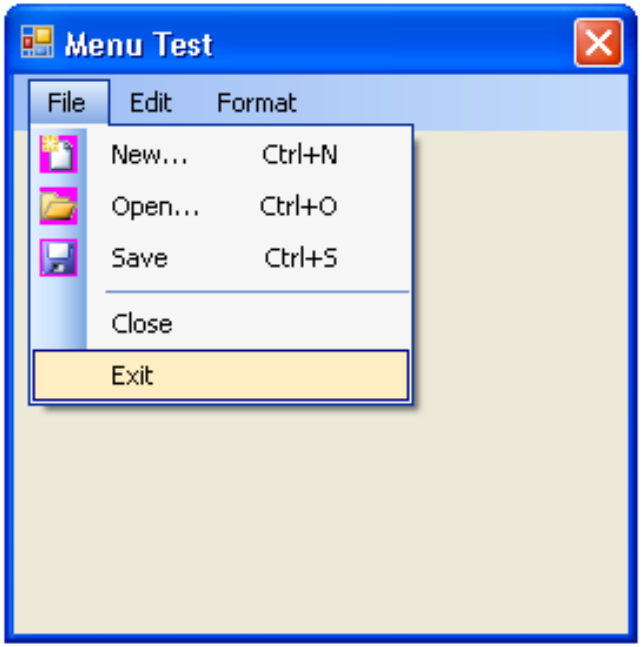
- Common Properties
  - **`Name`**: Indicates the name used in code to identify the menu. Usually set mnu+text (e.g. mnuFile, mnuNew, mnuOpen,...).
  - **`Checked`**: Whether menu item appears checked. Default **`false`**, meaning that the menu item is not checked.
  - **`Shortcut`**: Shortcut key for the menu item (i.e. *Ctrl + F9* can be equivalent to clicking a specific item).

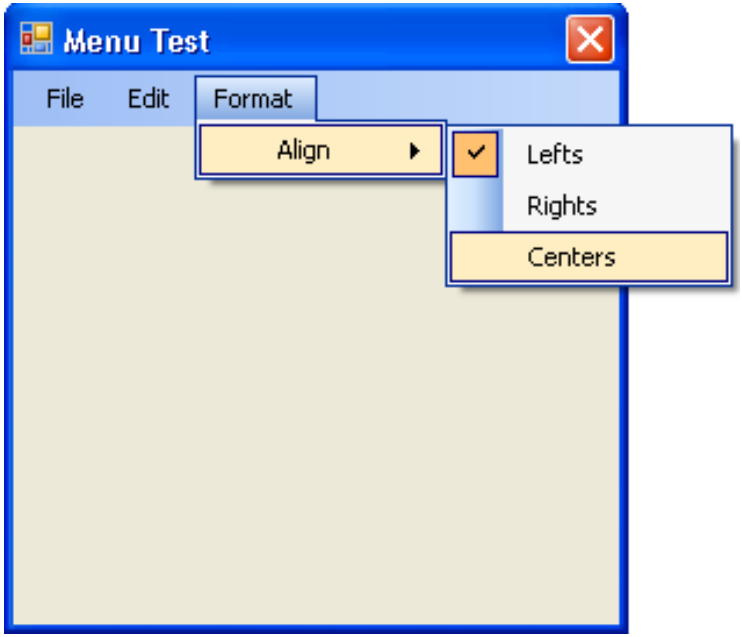
MenuStrip

- Common Properties
  - **`ShowShortcut`**: If **`true`**, shortcut key shown beside menu item text. Default **`true`**.
  - **`Text`**: Text to appear on menu item. To make an *Alt* access shortcut, precede a character with **`&`** (i.e. **`&File`** for **File**).
- Common Events
  - **`Click`**: Occurs when item is clicked or shortcut key is used. Default when double-clicked in designer.

MenuStrip

- Exercise





- Note: name = mnu+text (e.g. mnuFile, mnuNew)

ContextMenuStrip

- Represents shortcut menus that are displayed when the user clicks the right mouse button over a control or area of the form.
- To display shortcut menu of the control, sets the ContextMenuStrip property of the control to the name of **ContextMenuStrip** menu.

ToolBar

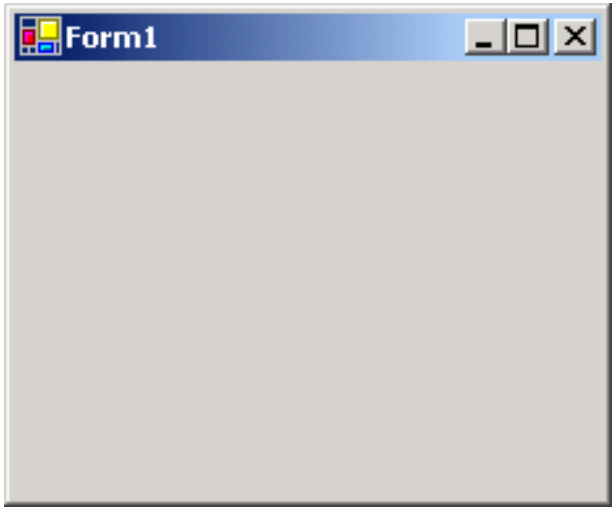
- The **`ToolStrip`** creates toolbars
- To add standard toolbars in the designer
  - Create a **`ToolStrip`** control.
  - In the upper right corner of the **`ToolStrip`**, click the smart task arrow to display the **`ToolStrip Tasks`** pane.
  - In the **`ToolStrip Tasks`** pane, choose **`Insert Standard Items`**
- To display a ToolTip
  - Set the **ShowItemToolTips** property of the control to **true**.

MDI Windows

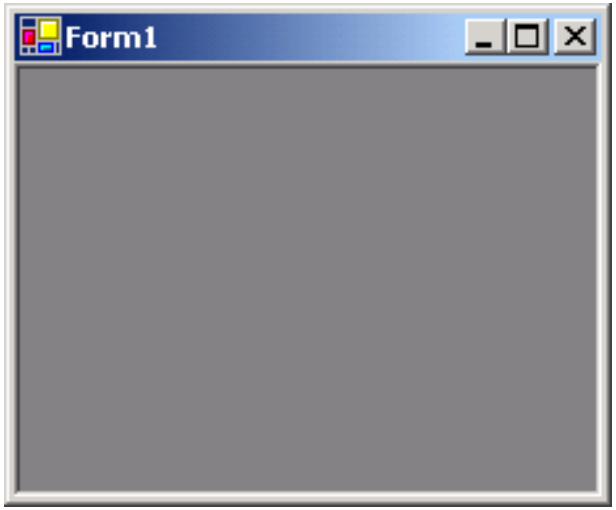
- Users can edit multiple documents at once
- Usually more complex then single-document interface applications
- Application window called parent, others child
- Parent and child menus can be merged
  - Based on **`MergeOrder`** property

MDI Windows

- SDI and MDI Forms



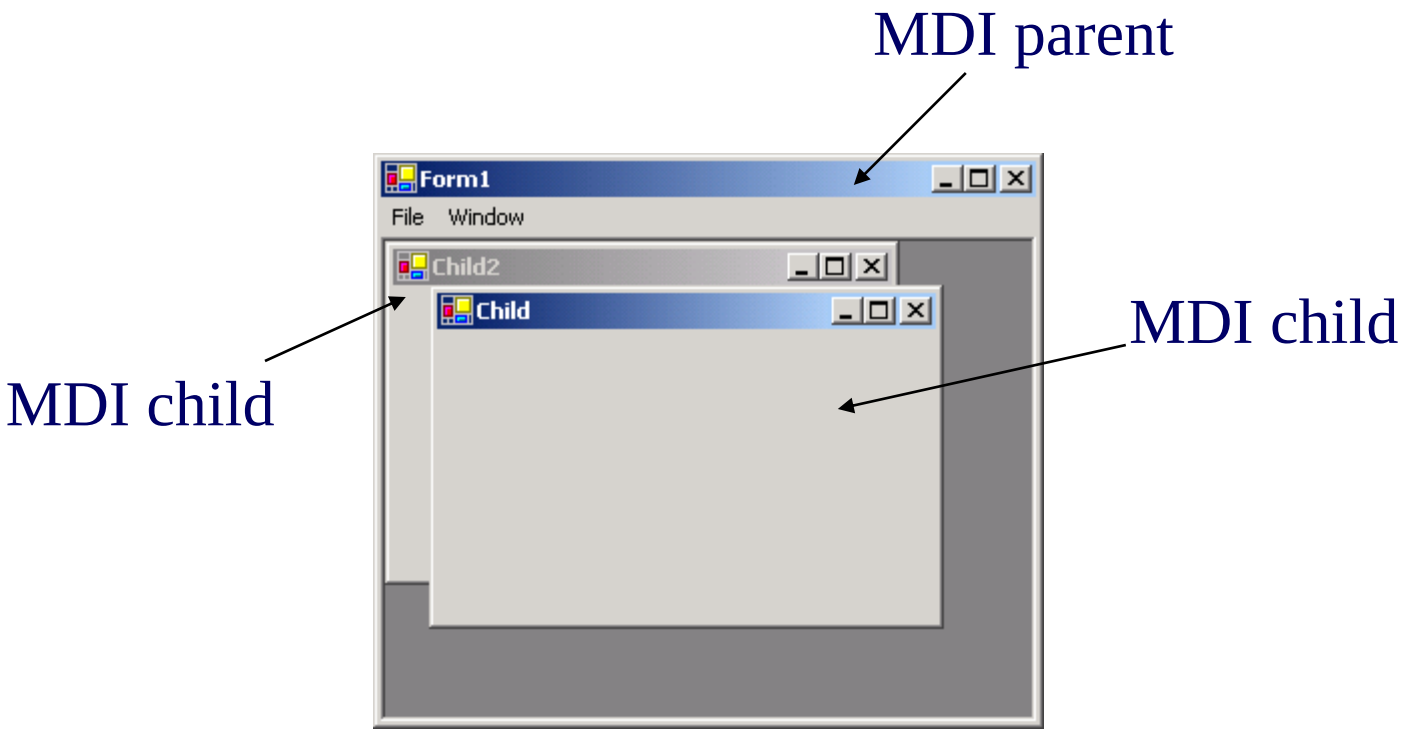
Single Document Interface (SDI)



Multiple Document Interface (MDI)

MDI Windows

- An example of MDI Parent and MDI Child



MDI Windows

- Child windows can be arranged in parent window:
  - **`Tiled windows`**: Completely fill parent, no overlap
    - Either horizontal or vertical
  - **`Cascaded windows`**: Overlap, same size, display title bar
  - **`ArrangeIcons`**: Arranges icons for minimized windows

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).

MDI Windows

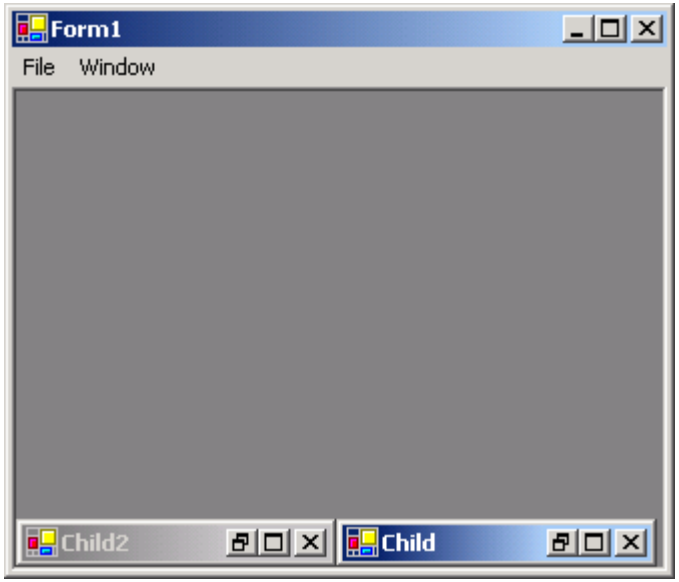
- Common MDI Parent Properties
  - **`IsMdiContainer`**: Indicates whether a **`Form`** can be an MDI. If **`True`**, the **`Form`** can be an MDI parent. Default is **`False`**.
  - **`MdiChildren`**: Returns the MDI children as an array of **`Form`**s.

MDI Windows

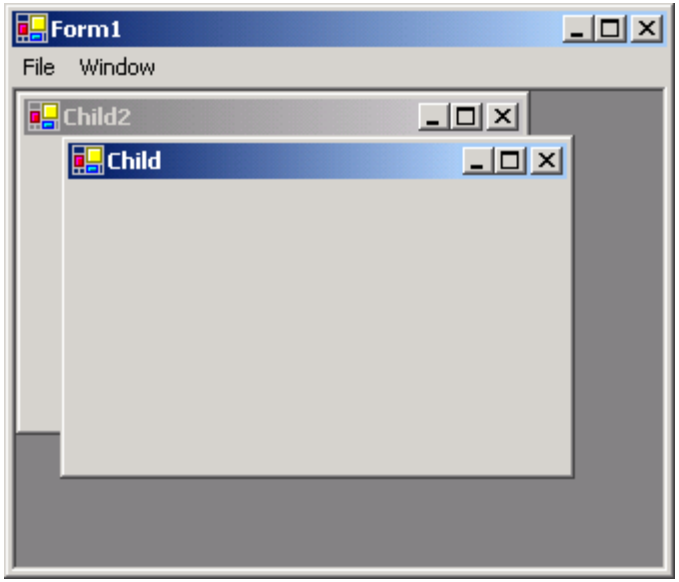
- Common Methods
  - **`LayoutMdi`**: Determines the display of child forms on an MDI parent. Takes as a parameter an **`MdiLayout`** enumeration with possible values **`ArrangeIcons`**, **`Cascade`**, **`TileHorizontal`** and **`TileVertical`**.
- Common Events
  - **`MdiChildActivate`**: Generated when an MDI child is closed or activated.

MDI Windows

- LayoutMdi enumeration values



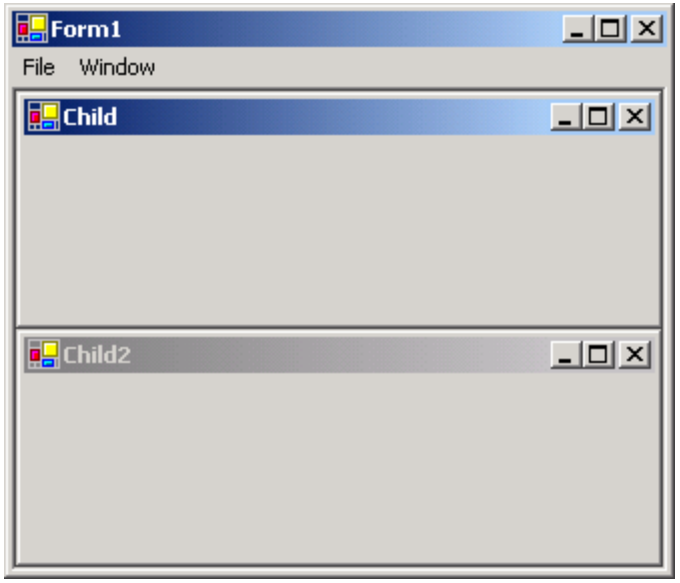
`ArrangeIcons`



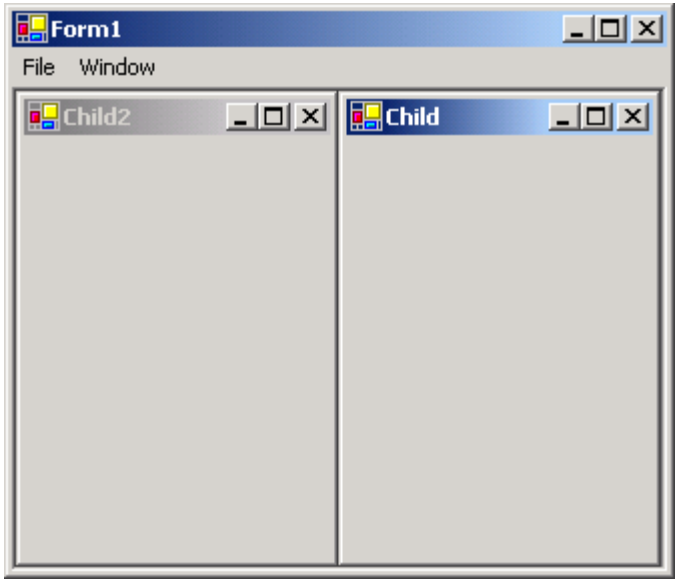
`Cascade`

MDI Windows

- LayoutMdi enumeration values



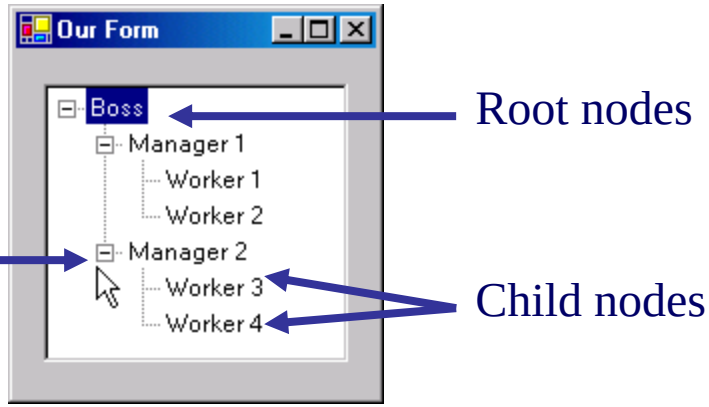
`TileHorizontal`

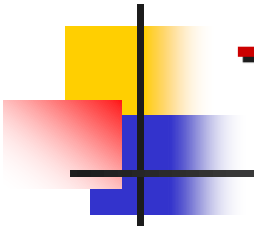


`TileVertical`

TreeViews

- Displays nodes hierarchically
- Parent nodes have children
- The first parent node is called the root
- Use Add method to add nodes





TreeViews

- Common Properties
  - **`CheckBoxes`**: Indicates whether checkboxes appear next to nodes. **`True`** displays checkboxes. Default is **`False`**.
  - **`ImageList`**: Indicates the **`ImageList`** used to display icons by the nodes. An ***`ImageList`*** is a collection that contains a number of **`Image`** objects.

TreeViews

- Common Properties
  - **`Nodes`**: Lists the collection of **`TreeNodes`** in the control. Contains methods **`Add`** (adds a **`TreeNode`** object), **`Clear`** (deletes the entire collection) and **`Remove`** (deletes a specific node). Removing a parent node deletes all its children.
  - **`SelectedNode`**: Currently selected node.
  - **`Checked`**: Indicates whether the **`TreeNode`** is checked. (**`CheckBoxes`** property must be set to **`True`** in parent **`TreeView`**.)

TreeViews

- Common Properties
  - **`FirstNode`**: Specifies the first node in the **`Nodes`** collection (i.e., first child in tree).
  - **`FullPath`**: Indicates the path of the node, starting at the root of the tree.
  - **`ImageIndex`**: Specifies the index of the image to be shown when the node is deselected.
  - **`LastNode`**: Specifies the last node in the **`Nodes`** collection (i.e., last child in tree).

TreeViews

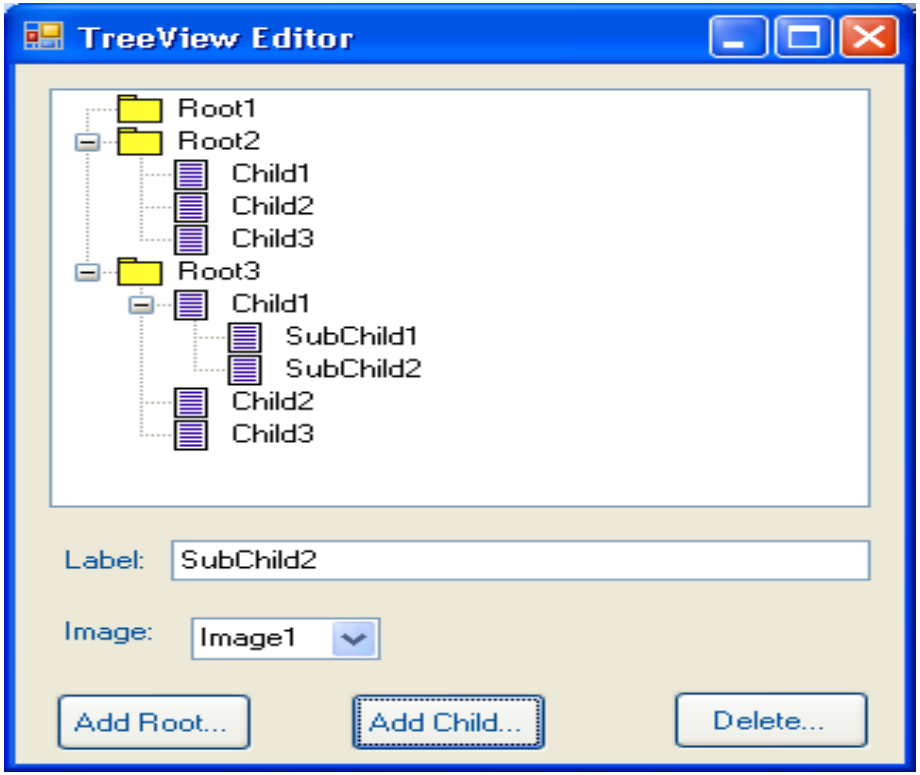
- Common Properties
  - **`NextNode`**: Next sibling node.
  - **`PrevNode`**: Indicates the previous sibling node.
  - **`SelectedImageIndex`**: Specifies the index of the image to use when the node is selected.
  - **`Text`**: Specifies the text to display in the **`TreeView`**.

TreeViews

- Common Methods
  - **`Collapse`**: Collapses a node.
  - **`Expand`**: Expands a node.
  - **`ExpandAll`**: Expands all the children of a node.
  - **`GetNodeCount`**: Returns the number of child nodes.
- Common Events
  - **`AfterSelect`**: Generated after selected node changes. Default when double-clicked in designer.

TreeViews

- TreeNode Editor Exercise



TreeViews

- Exercise
  - Set name properties
    - `treeView1`
    - `txtInput`
    - `comboBox1`
    - `btnAddRoot`
    - `btnAddChild`
    - `btnDelete`
  - Declare in class
    - `private TreeNode currentNode;`

TreeViews

- Set imageList1 property
  - **`ImageCollection`**: The images stored in this ImageList.
- Load Event

```
private void AddTreeView_Load(object sender, EventArgs e)
        {
            treeView1.ImageList = imageList1;
            comboBox1.Items.Add("Image1");
            comboBox1.Items.Add("Image2");
            comboBox1.SelectedIndex = 0;
        }
```

TreeViews

- btnAddRoot Click Event

```
private void btnAddRoot_Click(object sender, EventArgs e)
        {
            if (txtInput.Text.Trim() == "") return;
            TreeNode childNode = new TreeNode();
            childNode.Text = txtInput.Text;
            childNode.ImageIndex = comboBox1.SelectedIndex;
            treeView1.Nodes.Add(childNode);
        }
```

- treeView1_AfterSelect Click Event

```
private void treeView1_AfterSelect(object sender,
                                   TreeViewEventArgs e)
        {
            currentNode = e.Node;
        }
```

TreeViews

- btnAddChild and btnDelete Click Event

```
private void btnAddChild_Click(object sender, EventArgs e)
        {
            if (txtInput.Text.Trim() == "") return;
            TreeNode childNode = new TreeNode();
            childNode.Text = txtInput.Text;
            childNode.ImageIndex = comboBox1.SelectedIndex;
            currentNode.Nodes.Add(childNode);
            currentNode.ExpandAll();
        }
```

```
private void btnDelete_Click(object sender, EventArgs e)
        {
            currentNode.Remove();
        }
```

ListViews

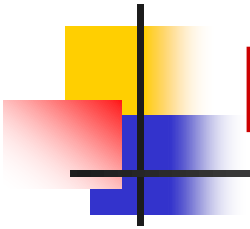
- Displays list of items
  - Can select one or more items from list
  - Displays icons to go along with items
- Common Properties
  - **`Activation`**: Determines how the user activates an item. This property takes a value in the **`ItemActivation`** enumeration. Possible values are **`OneClick`** (single-click activation), **`TwoClick`** (double-click activation, item changes color when selected) and **`Standard`** (double-click activation).

ListViews

- Common Properties
  - **`CheckBoxes`**: Indicates whether items appear with checkboxes. **`True`** displays checkboxes. Default is **`False`**.
  - **`Columns`**: The columns shown in detail view.
  - **`FullRowSelect`**: Indicates whether all SubItems are hightlighted along with the Item when selected.
  - **`GridLines`**: Display grid lines around all Items and SubItems. Only shown when in **`Details`** view.

ListViews

- Common Properties
  - **`Items`**: Returns the collection of **`ListViewItem`**s in the control.
  - **`LargeImageList`**: Indicates the **`ImageList`** used when displaying large icons.
  - **`MultiSelect`**: Determines whether multiple selection is allowed. Default is **`True`**, which enables multiple selection.
  - **`SelectedItems`**: Lists the collection of currently selected items.



ListViews

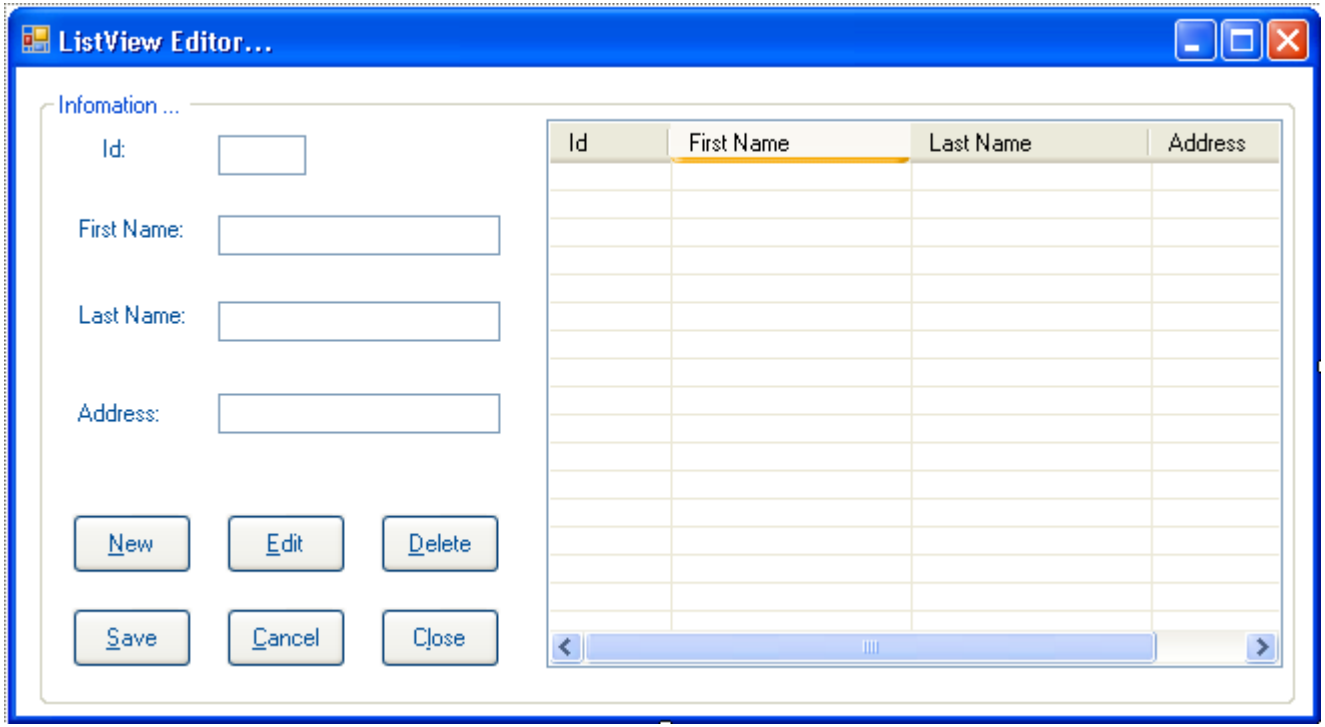
- Common Properties
  - **`SmallImageList`**: Specifies the **`ImageList`** used when displaying small icons.
  - **`View`**: Determines appearance of **`ListViewItem`**s. Values **`LargeIcon`** (large icon displayed, items can be in multiple columns), **`SmallIcon`** (small icon displayed), **`List`** (small icons displayed, items appear in a single column) and **`Details`** (like **`List`**, but multiple columns of information can be displayed per item).

ListViews

- Common Method
  - **`Add`**: Adds an item to the collection of items.
  - **`Clear`**: Removes all items from the collection.
  - **`Remove`**: Removes the specified item from the collection.
  - **`RemoveAt`**: Removes the item at the specified index within the collection.
- Common Events
  - **`ItemSelectionChanged`**: Occurs when the selection state of an item changes.

ListViews

- ListView Editor Exercise
  - Design the form



ListViews

- Exercise
  - Set name properties
    - txtId, txtFirstName, txtLastName, txtAddress
    - btnNew, btnEdit, btnDelete, btnSave, btnCancel.
  - Set listView1 properties
    - **`Columns`**: Add four columns
    - **`FullRowSelect`**: true
    - **`GridLines`**: true
    - **`MultiSelect`**: false
    - **`View`**: Details

ListViews

- Declare
  - `private bool modeNew;`
  - `private int row;`

- Add the function to enable or to disable the textboxes and buttons.

```
private void SetControls(bool edit)
    {
      txtId.Enabled = false;
      txtFirstName.Enabled = edit;
      txtLastName.Enabled = edit;
      txtAddress.Enabled = edit;
      btnNew.Enabled = !edit;
      btnEdit.Enabled = !edit;
      btnDelete.Enabled = !edit;
      btnSave.Enabled = edit;
      btnCancel.Enabled = edit;
    }
```

ListViews

- Load Event
  - `SetControls(false);`
- btnNew Click Event

```
private void btnNew_Click(object sender, EventArgs e)
        {
            modeNew = true;
            SetControls(true);
            row = listView1.Items.Count;
            txtId.Text = Convert.ToString(row+1);
            txtFirstName.Clear();
            txtLastName.Clear();
            txtAddress.Clear();
            txtFirstName.Focus();
        }
```

ListViews

- btnEdit Click Event

```
private void btnEdit_Click(object sender, EventArgs e)
        {
            modeNew = false;
            SetControls(true);
            txtFirstName.Focus();
        }
```

- btnCancel Click Event

```
private void btnCancel_Click(object sender, EventArgs e)
        {
            SetControls(false);
        }
```

ListViews

- btnDelete and btnClose Click Event

```
private void btnDelete_Click(object sender, EventArgs e)
        {
            try
            {
                listView1.Items.RemoveAt(row);
            }
            catch (Exception)
            {
            }
        }
```

```
private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }
```

ListViews

- btnSave Click Event

```
private void btnSave_Click(object sender, EventArgs e)
 {
   if (modeNew)
    {
      listView1.Items.Add(txtId.Text);
      listView1.Items[row].SubItems.Add(txtFirstName.Text);
      listView1.Items[row].SubItems.Add(txtLastName.Text);
      listView1.Items[row].SubItems.Add(txtAddress.Text);
    }
   else
    {
      listView1.Items[row].SubItems[1].Text = txtFirstName.Text;
      listView1.Items[row].SubItems[2].Text = txtLastName.Text;
      listView1.Items[row].SubItems[3].Text = txtAddress.Text;
    }
  SetControls(false);
 }
```

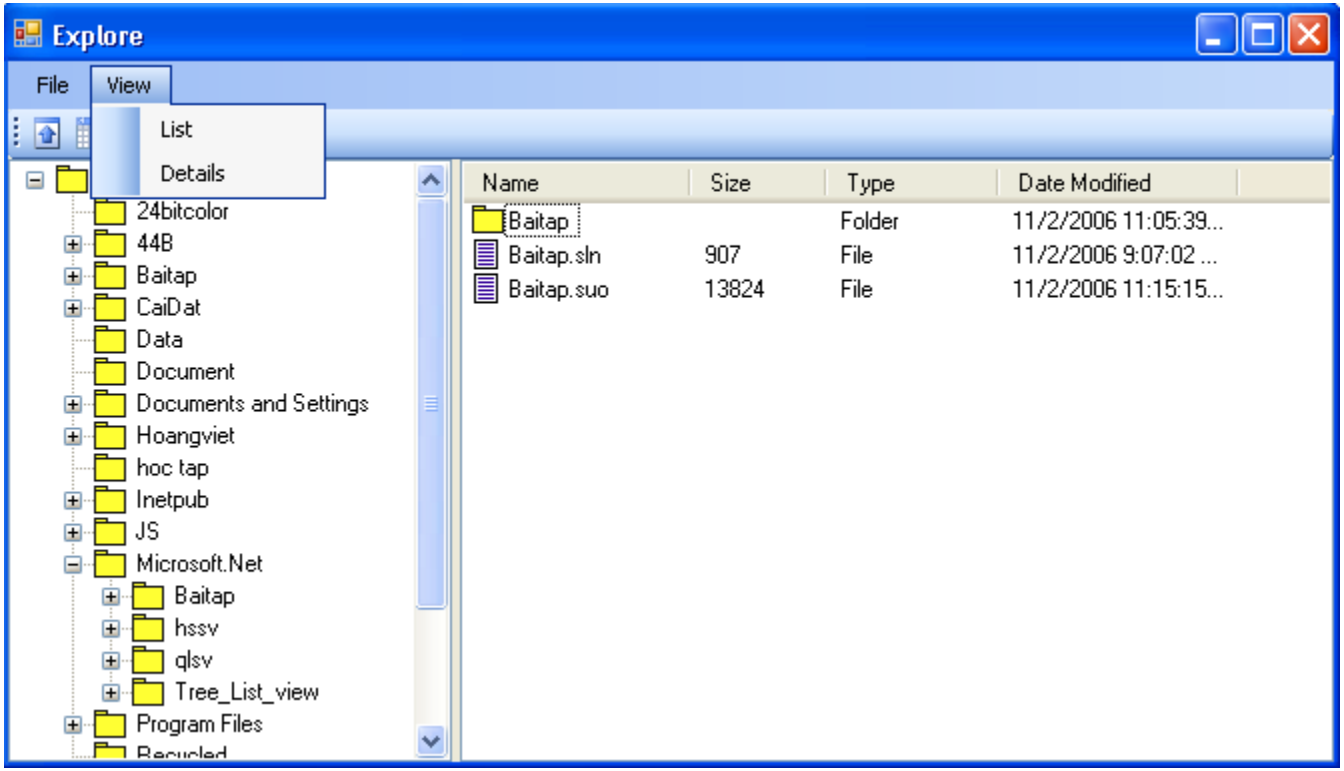
ListViews

- ItemSelectionChanged Event

```
private void listView1_ItemSelectionChanged(object sender,
                        ListViewItemSelectionChangedEventArgs e)
   {
      row = e.ItemIndex;
      txtId.Text = listView1.Items[row].SubItems[0].Text;
      txtFirstName.Text = listView1.Items[row].SubItems[1].Text;
      txtLastName.Text  = listView1.Items[row].SubItems[2].Text;
      txtAddress.Text   = listView1.Items[row].SubItems[3].Text;
   }
```

Homework

- Design Explore program



Homework

- Load and treeView1 Cick Event

```
private void myTreeView_Load(object sender, EventArgs e)
    {
      // add c:\ drive to treeView1 and  insert its subfolders
      treeView1.Nodes.Add("C:\\");
      LoadTreeView("C:\\",treeView1.Nodes[0]);
    }
```

```
private void treeView1_AfterSelect(object sender,
                                   TreeViewEventArgs e)
    {
       string currentDirectory = e.Node.FullPath;
       LoadFilesInDirectory(currentDirectory);
    }
```

Homework

- Load directories to treeView1

```
public void LoadTreeView(string dirValue, TreeNode parentNode)
   {
    string[] dirArray = Directory.GetDirectories(dirValue);
    if (dirArray.Length != 0)
     {
       foreach (string directory in dirArray)
       {
        DirectoryInfo curDirectory = new DirectoryInfo(directory);
        TreeNode myNode = new TreeNode(curDirectory.Name);
        parentNode.Nodes.Add(myNode);
        // recursively populate every subdirectory
        LoadTreeView(directory, myNode);
       }
     }
   }
```

Homework

- Load file to listView1

```
public void LoadFilesInDirectory(string curDirectory)
{
 // load directory information and display
 listView1.Items.Clear();
 DirectoryInfo newDirectory = new DirectoryInfo(curDirectory);
 // put files and directories into arrays
 DirectoryInfo[] dirArray = newDirectory.GetDirectories();
 FileInfo[] fileArray = newDirectory.GetFiles();
 // add directory names to ListView
 foreach (DirectoryInfo dir in dirArray)
    {
    ListViewItem newDirectoryItem = listView1.Items.Add(dir.Name);
    newDirectoryItem.SubItems.Add("");
```

Homework

- Load file to listView1

```
    newDirectoryItem.SubItems.Add("Folder");
    newDirectoryItem.SubItems.Add(dir.LastWriteTime.ToString());
    newDirectoryItem.ImageIndex = 0;
   }
   // add file names to ListView
   foreach (FileInfo file in fileArray)
    {
      ListViewItem newFileItem = listView1.Items.Add(file.Name);
      newFileItem.SubItems.Add(file.Length.ToString());
      newFileItem.SubItems.Add("File");
      newFileItem.SubItems.Add(file.LastWriteTime.ToString());
      newFileItem.ImageIndex = 1;
    }
}
```

Homework

- Other functions

```
private void mnuList_Click(object sender, EventArgs e)
    {
        listView1.View = View.List;
    }
```

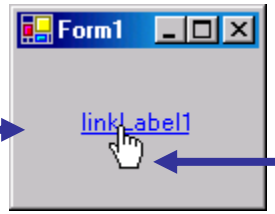
```
private void mnuDetails_Click(object sender, EventArgs e)
    {
        listView1.View = View.Details;
    }
```

```
private void tbUp_Click(object sender, EventArgs e)
    {
        string currentDirectory;
        currentDirectory = treeView1.SelectedNode.Parent.FullPath;
        LoadFilesInDirectory(currentDirectory);
    }
```

LinkLabels

- Displays links to other objects
  - Uses **`event handlers`** to link to right file or program
  - **`Start`** method of **`Process`** class opens other programs
- Derived from class **`Label`**, inherits functionality

`LinkLabel` on a form



Hand image displayed when mouse cursor over a **`LinkLabel`**

LinkLabels

- Common Properties
  - **`ActiveLinkColor`**: Specifies the color of the active link when clicked. Default is red.
  - **`LinkArea`**: Specifies which portion of text in the **`LinkLabel`** is treated as part of the link.
  - **`LinkBehavior`**: Specifies the link's behavior, such as how the link appears when the mouse is placed over it.
  - **`LinkColor`**: Specifies the original color of all links before they have been visited. Default is blue.

LinkLabels

- Common Properties
  - **`Links`**: Lists the **`LinkLabel.Link`** objects, which are the links contained in the **`LinkLabel`**.
  - **`LinkVisited`**: If **`True`**, link appears as if it were visited (its color is changed to that specified by property **`VisitedLinkColor`**). Default **`False`**.
  - **`Text`**: Specifies the text to appear on the control.
  - **`UseMnemonic`**: If **`True`**, **`&`** character in **`Text`** property acts as a shortcut (similar to the *Alt* shortcut in menus).

LinkLabels

- Common Properties
  - **`VisitedLinkColor`**: Specifies the color of visited links. Default is **`Color.Purple`**.
- Common Events
  - **`LinkClicked`**: Generated when link is clicked. Default when control is double-clicked in designer.