**Windows Controls**

- Windows created by a parent window
- An application uses them in conjunction with parent
- Normally used for simple I/O tasks
- Have a look and feel consistent with other application Windows
- Properties, appearance, behavior determined by predefined Control Class definitions
  - But behavior can be customized
  - Easy to set them up as common Windows objects
    - buttons, scroll bars, etc.
- Can also define custom Child Window Controls

**Child Window Controls**

- Windows created by a parent window
- An application uses them in conjunction with parent
- Normally used for simple I/O tasks
- Have a look and feel consistent with other application Windows
- Properties, appearance, behavior determined by predefined Control Class definitions
  - But behavior can be customized
  - Easy to set them up as common Windows objects
    - buttons, scroll bars, etc.
- Can also define custom Child Window Controls

**Some .NET Control Classes**

- Button
- Label (Static)
- GroupBox
- Panel
- CheckBox
- RadioButton
- HScrollBar
- VScrollBar
- TextBox (Edit)
- PictureBox
- ListBox
- ComboBox
- StatusBar
- TabControl
- ToolBar
- ToolTip
- CheckedListBox
- DataGrid
- DataGridTextBox
- DateTimePicker
- LinkLabel
- ListView
- MonthCalendar
- NumericUpDown - spinner buttons
- ProgressBar
- PropertyGrid
- RichTextBox
- TrackBar
- TreeView
- Others

**Creating a Control in .NET**

- To create a control and make it appear on a form:
  1. Declare and instantiate the control class
     ```csharp
     Button myButton;
     myButton = new Button();
     ```
  2. Initialize the control by setting its properties
     ```csharp
     myButton.Location = new Point(10, 10);
     myButton.Text = "Click Me";
     myButton.BackColor = Color.Red;
     ```
     - etc.
  3. Attach the control to the form (add to parent’s collection of controls) …
Attaching Controls to a Parent Form

• Assume we want to add myButton and myLabel controls to “this” form
• Three ways of doing it (assume we’ve instantiated the controls myButton and myLabel):
  1. myButton.Parent = this;
     myLabel.Parent = this;
  2. this.Controls.Add(myButton);
     this.Controls.Add(myLabel);
  3. this.Controls.AddRange(new Control[] { myButton, myLabel });

• Controls property: the collection of controls attached to the form
• #3 is done automatically by the Visual Studio Designer when you “drag” controls onto the form

Some Control Properties/Methods

• Common properties and methods
  – Derive from class Control
  – Text property
    • Specifies the text that appears on a control
  – TextAlign property
    • Alignment of text inside control
  – Focus() method
    • Transfers the input focus to a control
    • The control becomes the active control
  – TabIndex property
    • Order in which controls are given focus when user tabs
    • Automatically set by Visual Studio .NET Designer
  – Enabled property
    • Indicates a control’s accessibility and usability

Control Properties and Layout

• Visible property
  – Hide control from user
  • Or use method Hide()
• Anchor and Dock properties
  – Anchoring control to specific location
    • Constant distance from specified location
    • Default in Designer is Top-Left
  – Unanchored control moves relative to former position
  – Docking allows control to spread itself along an entire side
  • Both options refer to the parent container
• Size property
• BackColor, ForeColor properties
• Image, ImageAlign, BackgroundImage properties

Control Properties and Layout

Control Layout

Manipulating the Anchor property of a control.
Control Events
• All controls derive from System.Windows.Forms.Control
  – All inherit 50+ public events
  – Some common ones:
    | Event          | Event argument |
    |----------------|----------------|
    | Click          | EventArgs       |
    | DoubleClick    | EventArgs       |
    | ControlAdded   | ControlEventArgs|
    | ControlRemoved | ControlEventArgs|
    | Enter          | EventArgs       |
    | Leave          | EventArgs       |
    | Move           | EventArgs       |
    | Paint          | PaintEventArgs  |
    | Resize         | EventArgs       |
    | SizeChanged    | EventArgs       |
    | All other mouse events | MouseEventArgs |
• Event handling done as with Form events

Adding a Button Click Event Handler
• The Button Click Event Delegate is EventHandler()
  myButton.Click += new EventHandler(myButton_Click);
  ...
  private void myButton_Click(object sender, EventArgs e) {
    // Add handler code here
  }
• This code is inserted automatically when you use the Visual Studio Designer Properties Window to add a Click event handler
  – Or double click on the Control in Visual Studio Designer

Button Controls
• Rectangular objects, often with labels
• Intended to trigger an immediate action
  – Action is triggered by clicking mouse on button
  – Or pressing space bar if it has the input focus
• Some important Button properties:
  – Location, Size, BackColor, ForeColor, Cursor, Name, Text, TextAlign, Font, Image, ImageAlign, BackgroundImage, TabIndex,
  – Lots of others

Label Controls
• Controls designed for the display of static text
  – Called Static controls in Win32
  – User can’t change the text
• Can be changed in code
• Can also display graphics
• Have many of the same Properties as Buttons
• Can respond to events, but not really meant to do that

Button-Label Example Program
• Form has a Button control with Text: “Click Me”
• Form has a Label control that displays “Hello World” when button is clicked
  – In response to the button’s Click event
• Can be prepared manually from Visual Studio
  – Programmer must write code to instantiate the controls, attach them to the parent form, set up all their properties, and add the Button Click event handler
• Easier to use the Visual Studio Designer
  – Drag a button and label control from the toolbox to the form
  – Controls are automatically instantiated & “attached” to the form
  – Change the Properties of each in the Property window of each
  – Add the Button Click handler by double clicking on the button
  – Or using the Button’s Properties window (lightning bolt)
  – Add the following code in the skeleton handler
    label1.Text = “Hello World”;

Buttons with Images
• Button class has an Image Property
  – Set that property to display an image on background of the button
• Can be used in conjunction with Text Property
  – Text displayed on top of the image
• Make sure image fits in the button
  – Can use Image.GetThumbnailImage(…) to resize the image
    • Arguments: int w, int h, Image.GetThumbnailImageAbort gt, IntPtr p
  – Last two can specify a callback function & data – usually set to null and IntPtr 0, respectively
  – Returns the thumbnail image
  – This can be used as a general image resizing function
  – Alternatively, make the button be the size of the image
  – Change the button’s Size property
• Example Program: Button-Image
  – Does same as Button-Label, but now button has an image on it
GroupBox and Panel Controls

• Arrange components on a GUI
  – GroupBoxes can display a caption
    • Almost always contain other controls
      – Radio Buttons are very common
        • Only one active at a time
    • Text property determines its caption
  – Panels are used to group other controls against a background
    • Useful when you need a control that doesn’t do much
    • If contents of panel take up more space than panel itself, attached scrollbars can automatically appear
      • So user can view additional controls inside the Panel

GroupBox Control Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupBox</td>
<td>The controls that the GroupBox contains.</td>
</tr>
<tr>
<td>Common</td>
<td></td>
</tr>
<tr>
<td>Controls</td>
<td>The controls that the GroupBox contains.</td>
</tr>
<tr>
<td>Text</td>
<td>Text displayed on the top portion of the GroupBox (its caption).</td>
</tr>
</tbody>
</table>

Panel Control Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>Whether scrollbars appear when the Panel is too small to hold its controls. Default is false.</td>
</tr>
<tr>
<td>AutoScroll</td>
<td>BorderStyle</td>
</tr>
<tr>
<td></td>
<td>Border of the Panel (default: None; other options are Fixed3D and FixedSingle).</td>
</tr>
<tr>
<td>Controls</td>
<td>The controls that the Panel contains.</td>
</tr>
</tbody>
</table>

GroupBox-Panel Example Program

• Organizes one group of buttons in a GroupBox
  – GroupBox is labeled
• Organizes another group of buttons in a Panel that is too small to view its buttons
  – AutoScroll Property is set => Scroll bars automatically appear to permit user to view all the buttons inside the Panel
• Clicking any button causes a label control to indicate which button was clicked

Scroll Bars

• Used everywhere in GUIs
• Two purposes:
  – To shift (“scroll”) the visible area of a form/control
    • Scroll bar is attached to the control/form
    • Set parent form/control’s AutoScroll Property to true
  – To vary a parameter
    • Standalone scroll bar
• Scroll bar Properties that can be read/modified:
  – Size and Location on parent control/form
  – Range: Minimum and Maximum/Thumb position
  – Current Value of thumb position
  – Change values
    • SmallChange: Value change when user clicks on end arrows
    • LargeChange: Value change when user clicks on area between end arrows and thumb
ScrollBar Events

• Two events raised by ScrollBar controls
  – ValueChanged -- Data: EventArgs
    • Raised when Value property has changed, either by a Scroll event or programmatically
  – Scroll -- Data: ScrollEventArgs
    • Raised when scroll bar thumb has been moved, either by mouse or keyboard
    • Provides information about the event, including the new value and type of event
    • Scroll Event provides more information than ValueChanged
    • Some ScrollEventArgsProperties:
      – Int Value
      – ScrollEventType Type
        » Enumeration Members: SmallDecrement (L or T arrow), SmallIncrement(R or B), LargeDecrement(L or T arrows), LargeIncrement(R or B), ThumbTrack (Thumb down), ThumbPosition (thumb up), EndScroll (scroll operation done), Others

Scroll-Image Example

• Add standalone horizontal and vertical scrollbars to main form
  – Position horizontal one along bottom of form
  – Vertical one on right side, leaving space on right for 2 label controls
• Control the position of an Image with the scrollbars
• Label controls show current position (x,y) of image
• Events:
  – Paint: draw image in its new position
  – Scroll of horizontal scrollbar: set new x value of image position, change label1’s text to current scrollbar Value, & repaint
  – Scroll of vertical scrollbar: set new y value of image position, change label2’s text to current scrollbar Value, & repaint
  – Resize: reposition scrollbars and reset their Maximum values

Radio Buttons & Check Boxes

• Both are predefined “state” buttons that allow user to select or deselect a given option
  – Can be set to “on” or “off” (selected/unselected) state
  – For each, the Checked Property is set to true if button is unselected and true if selected
  – If AutoCheck property is true, state toggles when user clicks
• Radio Buttons
  – Almost always used in a group box from which only one button in the group can be selected at a time
  • Mutually exclusive options
  • They are all children of the group box … which is a child of the form
  – Displayed as little circles
  • Selected circle has a dot inside
• Check Boxes
  – If enclosed in a group box, any number of them can be selected
  – Displayed as little boxes
  • Selected boxes have check marks in them

Some CheckBox Properties and Events

Checkbox events and properties
<table>
<thead>
<tr>
<th>Description / Delegate and Event Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Properties</td>
</tr>
<tr>
<td>Unchecked: Whether or not the Checkbox has been checked.</td>
</tr>
<tr>
<td>CheckState: Whether the Checkbox is checked (contains a black checkmark) or unchecked (blank). An enumeration with values Checked, Unchecked or Indeterminate.</td>
</tr>
<tr>
<td>Text: Text displayed to the right of the Checkbox (called the label).</td>
</tr>
<tr>
<td>Common Events: (Delegate EventHandler, event arguments EventArgs)</td>
</tr>
<tr>
<td>UncheckedChanged: Raised every time the Checkbox is either checked or unchecked. Default event when this control is double clicked in the designer.</td>
</tr>
<tr>
<td>CheckStateChanged: Raised when the CheckState property changes.</td>
</tr>
</tbody>
</table>

Some RadioButton Properties & Events

RadioButton properties and events
<table>
<thead>
<tr>
<th>Description / Delegate and Event Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Properties</td>
</tr>
<tr>
<td>Checked: Whether the RadioButton is checked.</td>
</tr>
<tr>
<td>Text: Text displayed to the right of the RadioButton (called the label).</td>
</tr>
<tr>
<td>Common Events: (Delegate EventHandler, event arguments EventArgs)</td>
</tr>
<tr>
<td>Click: Raised when user clicks the control.</td>
</tr>
<tr>
<td>CheckedChanged: Raised every time the RadioButton is checked or unchecked. Default event when this control is double clicked in the designer.</td>
</tr>
</tbody>
</table>

Radio-Check Example Program

• Draws open or filled rectangles of different colors
• A ‘Color Selection’ group box containing radio buttons allows user to select a color
• A ‘Fill Rectangle’ check box determines whether the rectangle is filled or not