Windows 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

- Allow user to display/select data in standard ways
- Windows Environment does most of work in:
  - painting/updating a Control's screen area
  - determining what user is doing
- Can do the "dirty work" for the main window
- Often used as input devices for parent window
- Are the "working components" of Dialog Boxes
- Windows OS contains each control's "***WndProc***"
  - so messages to controls are processed in predefined ways
- Parent/child relationship with main window
  - Can have hierarchies of child windows
  - Parent and child communicate by sending/receiving messages
- Have been part of Windows since the first versions
- Roster has grown from six basic ones to an assortment of 20+ rich and varied 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 a Control class object

     ```
     Button myButton;
     myButton = new Button();
     ```

  2. Initialize the Control object by setting its properties

     ```
     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});
```

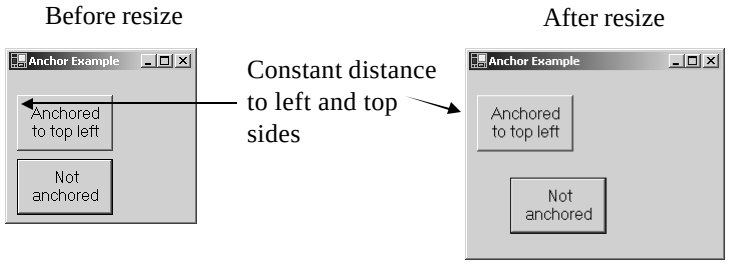
  - Controls property: the collection of controls attached to the form
  - # 3 is done automatically by the Visual Studio Designer when you "drag" controls onto the form

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
    - Indicate a control's accessibility and usability

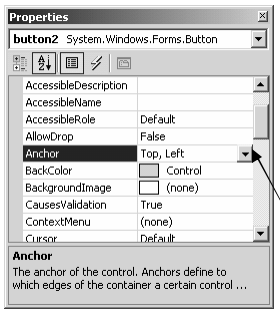
- Visible property
  - Hide control from user
    - Or use method Hide( )
- Anchor and Dock properties
  - Anchoring control to specific location
    - Constant distance from specified location
    - Default in Designer is Top-Left
  - Unanchored control moves relative to former position
  - Docking 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

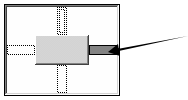
Control Properties and Layout



Anchoring demonstration.

Control Properties and Layout



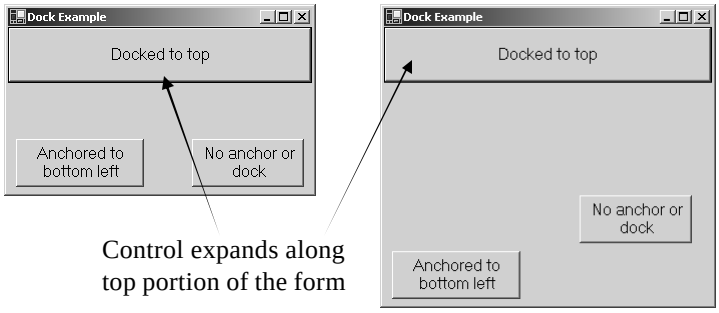


Darkened bar indicates to which wall control is anchored

Click down-arrow in Anchor property to display anchoring window

Manipulating the **Anchor** property of a control.

Control Layout



Control expands along top portion of the form

Docking demonstration.

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, System.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
  - **GroupBox**es 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
  - **Panel**s 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

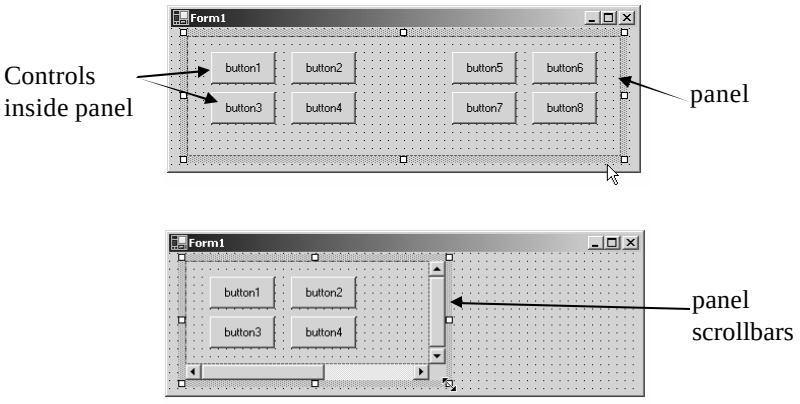
GroupBox Properties	Description
Common Properties	
Controls	The controls that the **GroupBox** contains.
Text	Text displayed on the top portion of the **GroupBox** (its caption).

Panel Control Properties

Panel Properties	Description
Common Properties	
AutoScroll	Whether scrollbars appear when the Panel is too small to hold its controls. Default is false.
BorderStyle	Border of the **Panel** (default **None**; other options are **Fixed3D** and **FixedSingle**).
Controls	The controls that the Panel contains.

Panel **properties.**

Panels



Creating a **Panel** with scrollbars.

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: Maximum and Minimum 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 scrollbar 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 ScrollEventArgs Properties:
      - Int Value
      - ScrollEventType Type
        - » Enumeration Members: SmallDecrement (L or T arrow), SmallIncrement(R or B), LargeDecrement (L or T areas), LargeInccrement(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 false 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	Description / Delegate and Event Arguments
Common Properties	
Checked	Whether or not the **CheckBox** has been checked.
CheckState	Whether the **Checkbox** is checked (contains a black checkmark) or unchecked (blank). An enumeration with values **Checked**, **Unchecked** or **Indeterminate**.
Text	Text displayed to the right of the **CheckBox** (called the label).
Common Events	*(Delegate **EventHandler, event arguments EventArgs**)*
CheckedChanged	Raised every time the **Checkbox** is either checked or unchecked. Default event when this control is double clicked in the designer.
CheckStateChanged	Raised when the **CheckState** property changes.

Some RadioButton Properties & Events

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

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