Windows Dialog Boxes, Text Boxes, and List Boxes

Dialog Boxes

- Popup child windows created by Windows
- Used for special-purpose input & output
  - A principal I/O mechanism in Windows
- Contain several child window controls
- Layout & what it does is are predefined
- In .NET they're just another Form
  - Derived from class Form
- We can design our own dialog boxes
- Five predefined "Common Dialog Boxes"

Types of Dialog Boxes

- Modal
- Modeless
- System Modal

Modal

- While visible, user can't switch back to parent window
  - (But user can change to other applications)
- User must explicitly end dialog box
  - Typically by clicking "OK" or "Cancel" buttons inside
- Most common type of dialog box
- Example: "About" box available with most Windows apps
- Message Boxes are simple Modal Dialog Boxes

System Modal

- A variety of modal dialog box
- With these user can't switch to other applications while dialog box is active
- A throwback to Win16

Modeless

- User can switch between dialog box and the parent window
- Used when dialog box must be visible while user interacts with the parent
- Example: dialog box resulting from "Find" or "Replace" menu item of many Windows applications

Common Dialog Boxes

- Predefined Modal Dialog Boxes that enable user to perform common I/O operations in a standard way
- Five of them -- all date back to Windows 3.1
  - FileDialog
    - Open/Save files in an easy and standard way
  - ColorDialog
    - Choose colors in an easy and standard way
  - FontDialog
    - Select fonts in an easy and standard way
  - PageSetupDialog
  - PrintDialog
    - Both related to printing
- Contained in classes derived from System.Windows.Forms.CommonDialog
- User interactions with common dialog box set properties that can be read & used afterwards

Using Common Dialog Boxes

1. Instantiate a common dialog object, e.g. ColorDialog:
   ```
   ColorDialog colordlg = new ColorDialog( );
   ```
2. Set its properties (optional)
   ```
   colordlg.Color = this.BackColor;
   ```
3. Call its ShowDialog( ) method to invoke the dialog box
   - Since Modal, execution halts until user dismisses the dialog box

   ```
   colordlg.ShowDialog( );
   ```
4. After it returns, use its properties changed by user actions
   ```
   this.BackColor = colordlg.Color;
   ```

– Almost always contain "OK" & "Cancel" buttons
  - "Abort", "Ignore", "No", "Retry', 'Yes" buttons are also defined
  - Button pressed by user is contained in ShowDialog return value
    - E.g., DialogResult.OK, DialogResult.Cancel, etc.

```
if (colordlg.ShowDialog( ) == DialogResult.OK)
    this.BackColor = colordlg.Color;
```

– Example program: Common-Color-Dialog
  - Note button inherits the new color

Common Font Dialog Box

- Allows the user to change fonts
- Class FontDialog
  - Properties:
    - Font font
    - Color Color
    - bool ShowColor
    - bool ShowEffects
    - bool ShowApply
    - bool ShowHelp
  - Instantiate and start with ShowDialog( ) member function just as for the Common Color dialog Box
- Example program: Common-Color-Font-Dialog

Using Visual Studio Designer to Create Common Dialog Boxes

- Just drag them from the toolbox onto the form
- Their properties can be accessed easily in their Properties Windows
- Still have to write code to invoke them
  - ShowDialog( )
- And code to use their changed properties

Programmer-Defined Dialog Boxes

- Define our own dialog boxes containing whatever controls are required for custom I/O
- Just define and instantiate a second class derived from class Form in the application
  - Encapsulates everything about the dialog box
  - Set desired properties
  - Add desired controls and event handlers
  - Start it with the object's ShowDialog() method
  - Main form's code stops executing until user dismisses the dialog box
    - DialogResult property returned by ShowDialog() will identify which button in dialog box was pressed to terminate it

Dialog-Manual Example Program

- Main form created with VS Designer as usual
  - Contains a "Start Dialog Box" button And a Label control
    - When user clicks the button, a modal dialog box with "OK" and "Cancel" buttons appears
    - The name of the button pressed by the user to dismiss the dialog box will be displayed in the main form's label control
  - The dialog box's buttons, properties, and button click handler methods are all defined in a second Form class
    - Handlers should set Dialog Box's DialogResult property
  - The second form class was coded manually
    - Easier to use Visual Studio to add the second dialog box class, set its properties, and add its button click handlers

Dialog-Designer Example Program

- Same functionality as Dialog-Manual application
- Add dialog box
  - With project name selected in Solution Explorer:
    - Select from main menu: Project | Add New Item | Windows Form
    - Or right click on project name and select Add | Windows Form
  - In either case the "Add New Item" dialog box comes up
    - Change the default name to SimpleDialogBox
  - VS Designer will create a new file containing the new class
  - As usual, add the "OK" & "Cancel" buttons to the new form by dragging them from the tool box
  - And add their click event handlers by double clicking on them or using the properties window (lightning bolt)
- Add "Start Dialog Box" button on main form
- And its click event handler as usual
  - In this handler add code to instantiate the dialog box, set its properties, and start it

Adding an Icon to the Dialog Box

- Set the form's Icon property
- One way:
  - this.Icon = new Icon("info.ico");
    - But this icon is in C:\Program Files\Microsoft Visual Studio 8\Common7\VS2005ImageLibrary\VS2005ImageLibrary\icons\Misc
      - Could give the complete path name
      - Or copy it to the project's debug directory
  - Better to include it as an embedded resource in the assembly
    - Visual Studio can do that
      - Go to form's properties box and click on the Icon Property's "Icon …" box
      - Navigate to the desired icon and select it

Getting Data from a Dialog Box

- Dialog boxes usually allow user to provide data for the application
- How to get data from the dialog box to the parent form:
  - Could use public fields (variables)
    - So other classes (the parent form) can access them
  - Better to use public properties
    - For protected access to private fields
  - Must be defined in the dialog box class
  - Properties with their get/set accessors can be coded manually
  - See DlgBoxPropertiesTest Example
    - Displays which of three buttons in a Dialog Box was pressed
    - Note use of this.Close() in Exit button handler to dismiss the Dialog Box

Radio-Check-Dialog Example

- Radio-Check application modified using a dialog box
  - Two classes:
    - ColorFillDialogBox class encapsulates a dialog box that allows the user to choose a color and fill option
      - Colors are shown in radio buttons in a "Color" group box
        - » Create and add the radio buttons in a loop
        - » Selected color (ColorRect) is a public Property added to the class get/set accessors index thru all controls in the Color groupbox
        - » Note use of Color.FromName(…) that creates a Color from a string
      - "Fill Rectangle" is a check box
        - » Check box state (Fill) is another public Property added to the class get/set accessors return/set the Checked state of the checkbox
    - Main Form1 class has a button to start the dialog box
      - Dialog Box's ColorRect and Fill Properties are used to change class-level variables colorRect and bFillRect after dialog box is dismissed
      - Paint event is forced
        - » Paint handler draws or fills a rectangle according to the values of colorRect and BFillRect

Modeless Dialog Boxes

- Stick around after invoked
- Start with Show() member method of DialogBox class
  - Not ShowDialog(), which starts it as a modal dialog box
  - We'll come back to these later

More Windows Controls

Text Input Controls

- TextBox
  - Formerly called an Edit Control
  - Allows user to type in text
    - Can be single line or multiline
- ListBox
  - Presents a scrollable list of selections for user to choose
- ComboBox
  - Combines the features of a Text Box and a List Box

Text Boxes

- Simplest is derived from TextBox class
  - RichTextBox class provides additional functionality
  - Both are derived from TextBoxBase class
    - Some Properties:
      - string Text
      - int MaxLength // max # of characters
      - int TextLength // (get only)
      - bool Multiline
      - string[ ] Lines // for multiline text boxes
      - int Lines.Length // # of lines
    - Most useful event:
      - TextChanged -- actually defined in Control parent class
      - Method: OnTextChanged()
      - Delegate: EventHandler
      - Argument: EventArgs

TextBox-Simple Example Program

- Creates a TextBox and a Label control
- Any time user changes text in the TextBox, it is reproduced in the Label control
  - Program handles the TextBox's TextChanged event
- Created with VS Designer
  - Just drag the TextBox and Label from the toolbox, change their properties, and add the TextChanged event handler

MultiLine Text Boxes

- Just set Multiline property to true
- Another property:
  - Lines
    - An array of strings that contains the text entered by user
    - Since it's an array, Lines also has a Length property
- Can add scrollbars
  - ScrollBars property:
    - None, Horizontal, Vertical, Both
    - For horizontal to work, WordWrap property must be set to false
- Give Notepad-like functionality
- Example: TextBox-Multiline

Non-textual Data in a TextBox

- Use Parse() method to convert Text property of a control to its numeric form before using it in a computation
- Each data type has its own Parse() method, e.g.:
  - int.Parse(); float.Parse(); decimal.Parse();
- Example – two text boxes contain string that represent numbers:
  - numTxtBox and priceTxtBox
  - To do computations need to convert to numeric values:

```
int num = int.Parse(numTxtBox.Text);                 //get number of items
decimal price = decimal.Parse(priceTxtBox.Text);  //get price per item
float totPrice = price*num;                          //compute total price
```

Formatting Data for Display

- Display numeric data in the Text property of a Label, Textbox, or Listbox
- Use ToString() and "format specifier codes"
  - Can format a numeric value to a string containing such features as: $, comma, decimal point, %
  - Also can specify # of digits to right of decimal point
    - xxx.ToString("format code")
- Some format codes (example: 1123.42817):
  - "C" currency $1,123.43
  - "F0" fixed point 1123
  - "F3 fixed point 1123.428
  - "N" number 1,123.43
  - "N3" number 1123.428

Compute-Interest Example

- Text Boxes for:
  - Principal, Interest Rate, Number of Years
- Labels for each
- Label for computed Total Interest
- Computes Total Interest:
  - Interest = Principal*Rate*Years
  - Note Parsing to get values from Text Boxes
  - And formatting to display result
  - Also note use of M or F suffix on numeric constants
    - M: decimal
    - F: float
    - C# defaults to double
- But what if user enters the wrong type of data?
  - Use a try/catch block
  - See ComputeInterestTryCatch example

List Boxes and Combo Boxes

- **List Box**
  - Contains lists of items that can be selected
  - Entire list is shown
  - User selects items
  - Selected item is highlighted
  - Encapsulated in class ListBox
- **Combo Box**
  - Text box combined with a list box
  - List box can be displayed at all times or pulled down
  - User selects item from list & item is copied to text box
  - One type allows user to type into text box
  - Encapsulated in class ComboBox
- For both, scroll bars are added automatically as needed

List Box "Items" Property

- The list of Items in a list box is a collection (like ArrayList)
  - These collections have methods that allow programmer to:
    - Add items, insert items, remove items, refer to individual items, count items, get selected item, & clear the collection
  - listBox1.Items.Add(ItemValue);
  - listBox1.Items.Insert(IndexPosition, ItemValue);
  - listBox1.Items.Remove(ItemValue);
  - listBox1.Items.RemoveAt(IndexPosition);
  - Referring to a given item:
    - listBox1.Items[IndexPosition];
  - Number of items in list
    - listBox1.Items.Count
  - SelectedIndex Property – stores index of item selected
    - int x = listBox1.SelectedIndex; // retrieve index of selected item
    - listBox1.SelectedIndex = 3; // select item 3 (will appear selected)
  - listBox1.Items.Clear( ); // remove all items from list

Using Designer to Fill a List Box at Design Time

- Select the List Box control on the form
- Scroll Properties window to "Items" property
- Click on "…" to open "String Collection Editor"
  - Type in the items in the list, ending each with Enter key
- Note in Designer Generated Code:
  - listBox1.Items.AddRange(new object[] {"str1", "str2", …});

ListBox-Simple Example

- Initial list box contents set at design time
- "Add Item" button allows user to add items to the list box using a text box
- "Get Current Selection" button displays currently-selected item from the list box in a label control

Combo Box

- Very Similar to a List Box
- Has an associated Text Box control
  - Text property is what is typed by user
  - Text property can be set in code
- DropDownStyle Property
  - Simple, DropDown