In-Class Worksheet #4

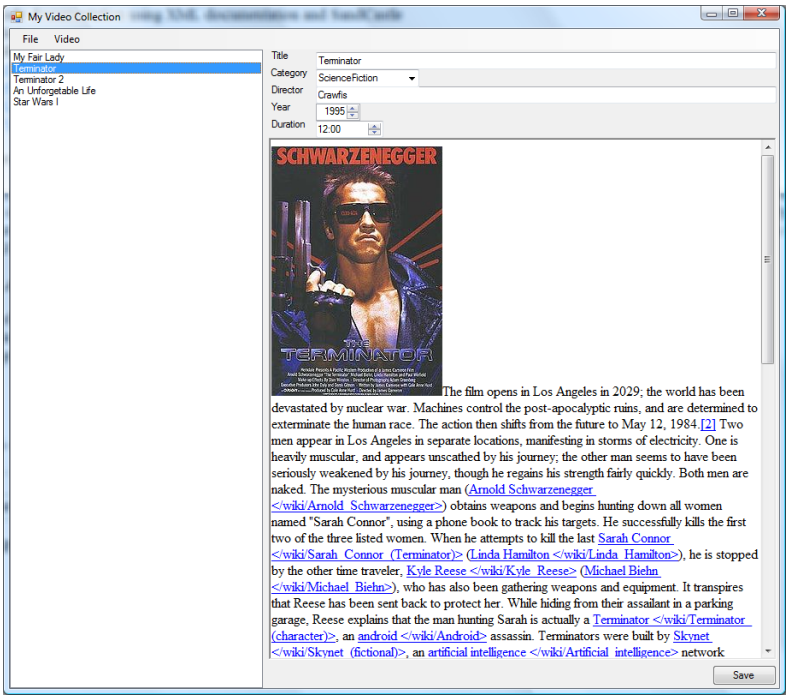
Creating a Windows Forms application with Data Binding

You should have Visual Studio 2008 open.

1. Create a new Project either from the shortcut on the Start page or through the file menu,
2. Select the type to be a C# Windows Forms
3. In the Location textbox, enter your desired hard drive location. Note, that this will be on the mounted z: drive in the CSE environment. Name the Project *Lastname*VideoCollection where *Lastname* is replaced with your name or an alias.
4. Select OK.

Create the VideoCollection application:

We are going to create a simple data entry system for movies or videos. My implementation is illustrated on the right.



5. Right-click on Form1.cs in the Solution Explorer and select Rename. Change Form1.cs to MyVideoForm.cs. You will be prompted to change the name throughout the solution. Click yes.
6. Change MyVideoForm's ***Text*** to something more meaningful for the title bar.
7. Drag a MenuStrip over to the form. Click the little Tasks icon and select Embed in ToolStrip Container. Note, in general you will only want to do this with Toolbars, but I want to demonstrate the ToolStrip Container. Make sure the GripStyle of the menu is set to visible. Add a File and a Video menu items. Under File, add "Open …", "Save Library …", and "Exit". Under Video, add "New Video". *Note: By convention (and user interface studies and research), we add ellipsis to the end of Menu names when the user will be prompted for more information. In this case, we are indicating to the user that Open and Save Library will prompt the user for additional information. Ideally, selecting a menu with an ellipsis should be free from any side effects. In other words, the user should be able to cancel the operation and return to the same state as before selecting the menu item.*
8. Drag a SplitContainer to the form. Change its Orientation to Vertical and position the splitter similar to the example.
9. Drag a ListBox to the left panel of the split container. Set its Dock property to Fill.
10. Drag a TableLayoutPanel to the right side of the panel. Give it 7 rows and 2 columns. Add labels to the first column for the first 5 rows as shown in the figure.
11. Add the following controls for the indicated label:
    - Title: TextBox
    - Category: ComboBox
    - Director: TextBox
    - Year: NumericUpDown. Set its minimum value to 1920 and maximum value to 2020.

- Duration: DateTimePicker. Set is Format to Custom, the CustomFormat to h:mm and the ShowUpDown to true.

12. Drag a Rich Text Box to column 1, Row 6 of the TableLayoutPanel. In the Properties window note that there are properties for this (and all of the controls add to the table) for the column and row. This should say column 0 and row 5. Set the ColumnSpan property to 2. This will have the control span across both columns. Set its Dock property to Fill. A rich text box is like a simple MS word viewer. It allows hyperlinks, images, text formatting, etc.
13. Drag a button to column 2 of the last row. Set its text to Save, rename the member variable to saveButton and set its Anchor to the Bottom-Right.
14. Ok, let's pretty that table up a little. Right-click in a grey area of the table and select "Edit Rows and Columns …". Set both Columns to AutoSize. Change the Combobox to Rows and set each row except row 6 to AutoSize. Set Row 6 to 100%.
15. One thing you should notice by now is that we have not written any code or done anything technically challenging. Why not relegate these tasks to someone more qualified (cheaper than a programmer?) in the look and feel or human factors of the design. This is precisely what WPF does with the separation of the XAML code from the C# behavior code.
16. OK, so let's start programming. But first, go to View->Other Windows->Document Outline. This window gives you a treeview of your controls. You can drag them around if you get the order wrong. Be advised that sometimes things get confused and you may have to undock and re-dock or play with things to get the system into the state you think it should be.
17. We are going to need two additional **components** , an OpenFileDialog and a SaveFileDialog, so drag each of those over to the form. Rename them to openLibraryDialog and saveLibraryDialog. Set the Filter property to "All Files (*.*)|*.*|Video Library (*.vid)|*.vid" without the quotation marks and no extra spaces. The format of this string is such that a display name is followed by a regular expression. These and additional pairs are separated by the character '|'. Set the Filter Index to 2 for both of these (defaulting to .vid files). Note, the .vid is my own concoction; you can make it anything you want.

The Video Data class

18. Add a new class to your project called VideoInfo. Add the enum Genre and the class as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace VideoLibrary
{
    [Serializable]
    public enum Genre { Drama, Comedy, Western, ScienceFiction, Horror, Family };

    [Serializable]
    public class VideoInfo
    {
        public VideoInfo()
        {
            Title = "Enter movie Title ...";
            Category = Genre.Comedy;
            Director = "Enter Director's full name";
            Year = 2010;
            Duration = new TimeSpan();
        }
        public string Title { get; set; }
        public Genre Category { get; set; }
        public string Director { get; set; }
```

```
        public int Year { get; set; }
        public TimeSpan Duration { get; set; }
        public string Review { get; set; }

        public override string ToString()
        {
            return this.Title;
        }
    }
}
```

19. The [Serializable] ***attribute*** decorates the class, letting the Type reflection system know that this data can be streamed to disk or across the network. We will see how easy it is to use the built-in .NET support for serialization to write and read instances of this class to disk.

MyVideoForm's Data

20. Add the following members to your MyVideoForm class:

```
List<VideoInfo> videoLib = new List<VideoInfo>();
BindingList<VideoInfo> bindingVideoLib;
```

21. Use the smart tag on BindingList to add the required namespace System.ComponentModel.
22. BindingList is a wrapper around List that allows communication with subscribers about the contents of the list changing. We will use its support to automatically inform the listbox that it should refresh its presentation of the data. Initialize the bindingVideoLib as follows (after the InitializeComponents call in the constructor):

```
bindingVideoLib = new BindingList<VideoInfo>(videoLib);
bindingVideoLib.RaiseListChangedEvents = true;
bindingVideoLib.AllowNew = true;
bindingVideoLib.AllowEdit = true;
```

23. Now, let's tie (bind) the data to the display. After the lines above add the following to set the data source for the listbox and to select what property to use to display:

```
this.listBox1.DataSource = bindingVideoLib;
this.listBox1.DisplayMember = "Title";
```

24. We also want to set-up the Combobox for the Genre. We can also use binding and a datasource for this. Since this data will never change, we do not need a special binding. Add the following line:

```
this.GenreComboBox.DataSource = Enum.GetValues(typeof(VideoLibrary.Genre));
```

Adding New Videos

25. Double-click the Video->New Video menu item and add the following code:

```
private void newVideoToolStripMenuItem_Click(object sender, EventArgs e)
{
    VideoInfo newVideo = new VideoInfo();
    bindingVideoLib.Add(newVideo);
    this.listBox1.SetSelected(videoLib.Count - 1, true);
}
```

26. This creates a new blank instance of VideoInfo and then adds it to the List. Note we use the bindingList to add it to the list. This ensures that the listbox is updated. As a nice feature, we also set the current selected index to this new entry so the user can immediately start typing in data. Without this last line, we would add an entry and then have to select it to start editing it.
27. Run and add new entries to your system. Note that we cannot change them now since we have not added that logic.

The Save Button

28. The Save button should collect the information in the controls and update the VideoInfo instance that is currently selected. Select the Save button, go to the events display in the Properties (the lightning bolt) and double click the space beside the Click event. This will add a saveButton_Click method to MyVideoForm.cs and take you to the code view within this method. Make it look like the following:

```
private void saveButton_Click(object sender, EventArgs e)
{
    int index = this.listBox1.SelectedIndex;
    VideoInfo selectedVideo = this.listBox1.SelectedItem as VideoInfo;
    if (selectedVideo != null)
    {
        selectedVideo.Title = this.titleTextBox.Text;
        selectedVideo.Director = this.directorTextBox.Text;
        selectedVideo.Category = (Genre)this.GenreComboBox.SelectedIndex;
        selectedVideo.Year = (int)this.numericUpDown1.Value;
        selectedVideo.Duration = new TimeSpan(this.dateTimePicker1.Value.Hour,
this.dateTimePicker1.Value.Minute, this.dateTimePicker1.Value.Second);
        selectedVideo.Review = this.richTextBox1.Rtf;
        bindingVideoLib.ResetItem(index);
    }
}
```

29. Note that the listBox1 has properties for selected index and selected item (as well as others) that allow you to get the entry selected.
30. The last line is the key to keeping the list view in sync. Run the program. Add a new entry, change the Title and click the Save button. Notice how the listbox changed its display when save was hit.

Keeping the two views in sync

31. OK, ideally we want is to display the VideoInfo data in our controls when an instance is selected in the listbox. To accomplish this we need to subscribe to the selectedindex changed event of the listbox and update all of the controls.
32. Add the following method to your Form:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    VideoInfo video = (VideoInfo) this.listBox1.SelectedItem;
    if (video != null)
    {
        this.titleTextBox.Text = video.Title;
        this.directorTextBox.Text = video.Director;
        this.GenreComboBox.SelectedIndex = (int)video.Category;
        this.numericUpDown1.Value = video.Year;
        this.dateTimePicker1.Value = new DateTime(2010,1,1,video.Duration.Hours,
video.Duration.Minutes, video.Duration.Seconds);
        this.richTextBox1.Rtf = video.Review;
    }
}
```

33. In the designer, select the listbox and then find the event for SelectedIndexChanged. Click the combo box beside this event and select the method you typed in above. Observe that you do not need to let visual studio create the routine for you. You can use a pre-existing one.

Saving and Opening Libraries with Serialization

34. Double click the Save Library menu item and add the following code:

```
private void saveLibraryToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (this.saveLibraryDialog.ShowDialog() == DialogResult.OK)
```

```
        {
            XmlSerializer s = new XmlSerializer(videoLib.GetType());
            using (TextWriter w = new StreamWriter(saveLibraryDialog.FileName))
            {
                s.Serialize(w, videoLib);
            }
        }
    }
```

35. This will display the SaveFileDialog and if the OK button was clicked will proceed to create the indicated file (you will be prompted about overwriting an existing file). The XmlSerializer has a serialize method that takes a stream to write into and an object that should be serialized. We are passing in the videoLib which is a List<VideoInfo>. Fortunately, the List<T> class is serializable and we indicated that VideoInfo is also serializable, so this will create an xml file (we called it a .vid file) with all of our entries in it.
36. Run the program, type in a few entries and then save the library. Go to the location you saved the file to and open it (using Notepad, WordPad, XML Notepad or Visual Studio). The last two will provide pretty printing. You can also rename it to an .xml file and then open it with Internet Explorer. The serializer used the property names of VideoInfo for the tags, a tag VideoInfo for each instance and then an ArrayOfVideoInfo tag for the root of the XML file.
37. Double click the Open menu item and add the following code:

```
    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (this.openLibraryDialog.ShowDialog() == DialogResult.OK)
        {
            XmlSerializer s = new XmlSerializer(videoLib.GetType());
            using (TextReader reader = new StreamReader(openLibraryDialog.FileName))
            {
                List<VideoInfo> newVideos = (List<VideoInfo>)s.Deserialize(reader);
                foreach( VideoInfo video in newVideos)
                    bindingVideoLib.Add(video);
            }
        }

    }
```

38. Since we already have things set-up. This really imports records into our system, by deserializing to a temporary List and then adding them to the master list.

Closing and Testing

39. Double-click the Exit menu button and type in **this**.Close(); Ideally, you might want to keep track of whether changes were made, and if so either prompt the user or do an automatic save before exiting.
40. That is a wrap! Run and test your application.
41. Bugs / Missing Features:
    - Clicking the Save button with no entries does not do anything. Adding a new entry then loses this work. The Controls should be gray'ed out until an entry is selected, or a new button should be added that creates a new video and initializes it to the data in the controls.
    - There is no validation, so a 10 hour movie is allowed.
    - For the most part resizing works, except when you make it real small (for a cell phone). Easily fixed with the minimum size in the Form properties. However, the Save button will be off of the form.
    - Work can be lost when Exit is selected.