Create First WPF Application

New Project

The first four short sections of this walk though introduce you to using the WPF Designer for creating, moving, aligning and sizing controls.

To create your first WPF Application Project

1. Start Visual Studio 2010.
2. From the File menu hover over New, then select New Project…

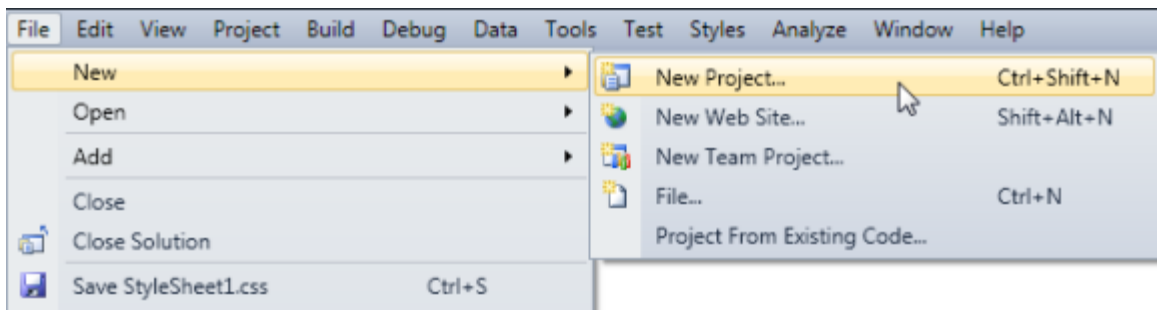

Figure 1 New Project Menu Selection

3. New Project Dialog will be displayed.

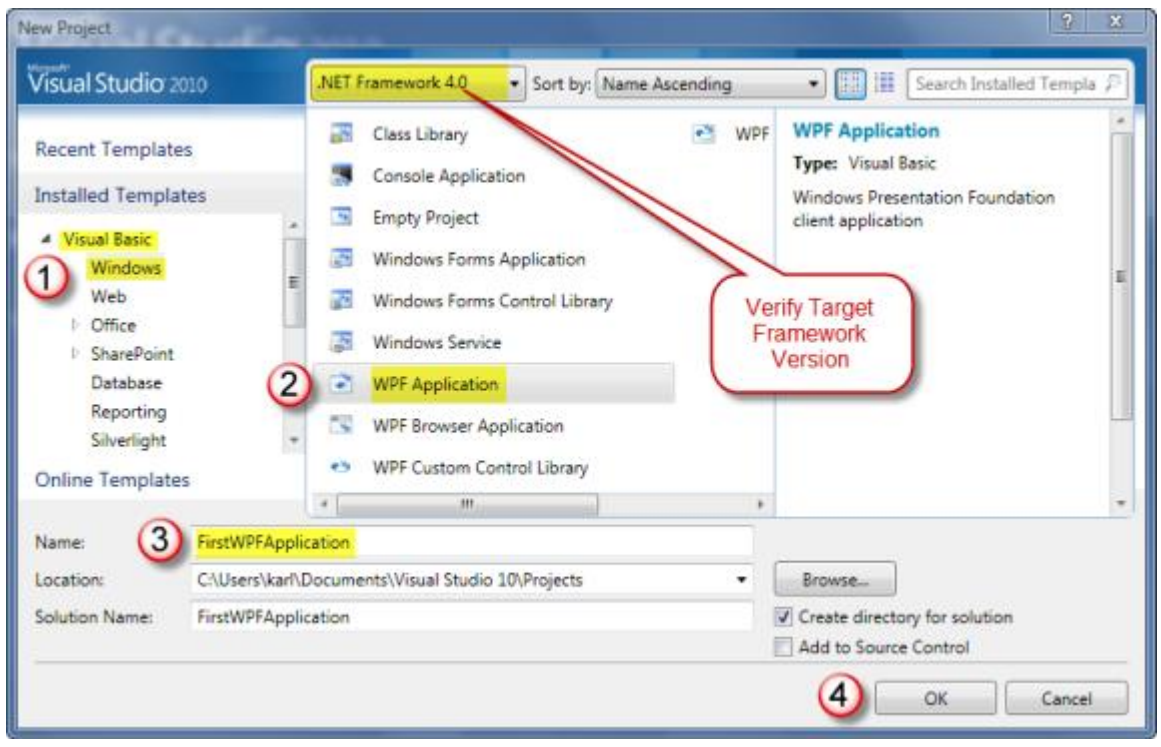

Figure 2 New Project Dialog

<u>Note:</u> The Target Framework Version ComboBox value defaults to the last selection made. When creating a new project, verify the desired Target Framework Version is selected.

- Step 1 – Select language; Visual Basic or C# then select Windows.
- Step 2 – Select WPF Application.
- Step 3 – Enter project name.

  <u>Tip:</u> While it is permissible to have spaces in the project name this practice discouraged.

- Step 4 – Click OK to create your new WPF Application.

Editing Window Title

Let's give our application a proper title. The Title property is displayed in the top left side of the window chrome.

1. Select the Window object by clicking on the Window border in the designer or by clicking the design surface outside of the window contents.
    1. Notice the Window object four corners now have resize adorners. These are used when resizing the Window object with the mouse.
    2. Notice the Window object's properties are now displayed in the Properties Window and that the Title property is selected.

       <u>Note:</u> Each control has a default property that is automatically selected in the Properties Window when that control is selected.

2. Change the Title property to "First WPF Application"

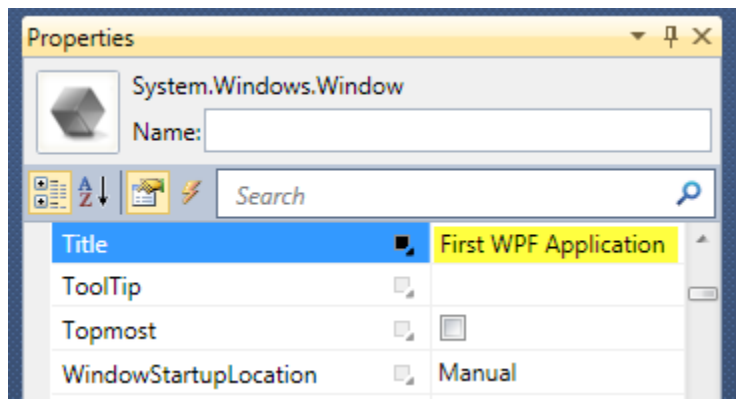

Figure 3 Properties Window

1. The designer keeps all related tool windows and view synchronized for your selection.

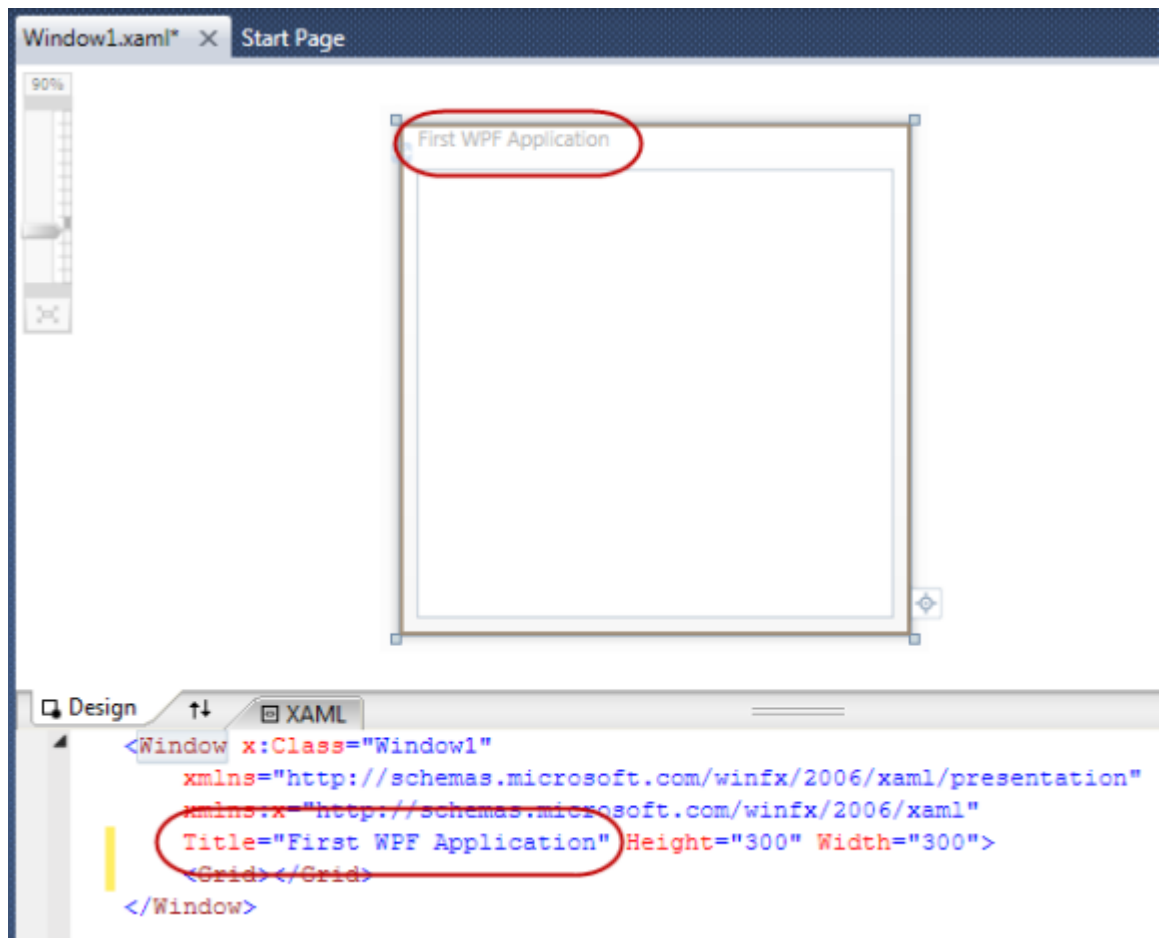

Figure 4 WPF Designer and XAML Editor

Creating Controls

Controls can be added to an application using the ToolBox or by editing the XAML text directly in the XAML Editor.

1. Several techniques can be used to add a control from the ToolBox.
    1. Select a control from the ToolBox by clicking it, then clicking the design surface where you want the control to be created.
    2. Select a control from the ToolBox by clicking it, and then draw the control on the design surface where you want the control to be created.
    3. Click and drag a control from the ToolBox to the design surface. See Figure 5 below.
    4. With the target control selected on the design surface, double click the control in the ToolBox.

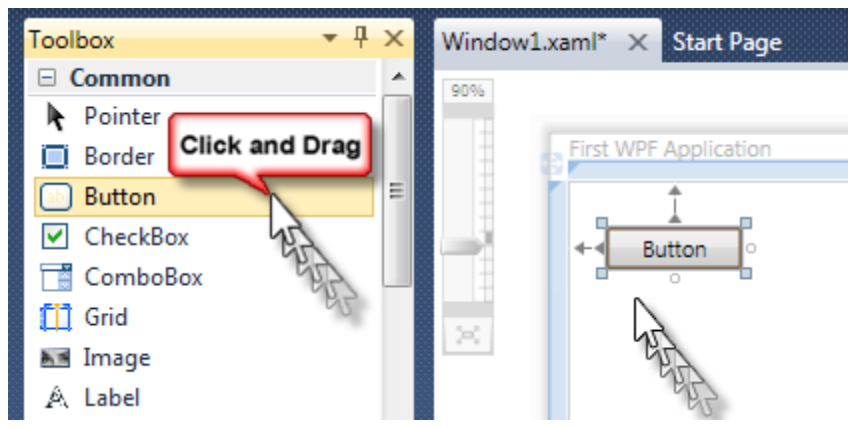

Figure 5 Create Control

    5. Practice creating a button control using the above techniques on the design surface. When done create a button in the position shown in Figure 5.
    6. Notice when you drag the button to the design surface, the Grid control's blue rail adorner is displayed.

Control Adorners and Snaplines

The WPF Designer displays control adorners on the design surface to assist developer when using the mouse to move or resize a control. Adorners also supply visual feedback about control alignment and other control specific information.

When controls are being moved or resized, the WPF Designer displays various types of snaplines to assist the developer in positioning a control on the design surface relative to other controls.

1. Alignment Adorners
    1. Select the button on the design surface.
    2. In the Properties Window, select Category View and scroll the Layout category into view as in Figure 6 below.
    3. The button in Figure 6 is aligned to the Top, Left. This is similar to Windows Forms Docking.
        1. The button's top edge is pinned 37 pixels down from the top of the Grid. (see second value in the Margin property below)
        2. The button's left edge is pinned 38 pixels from the Grid. (see first value in the Margin property below)

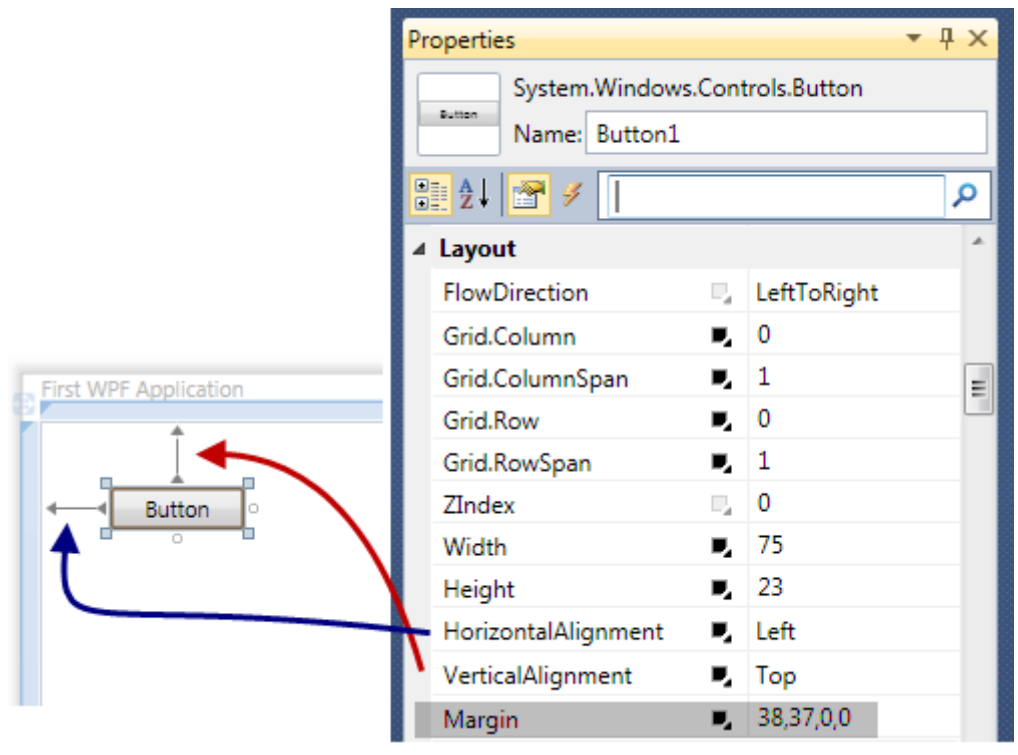

Figure 6 Alignment Adorners

Tip: *The Margin property is of type Thickness. The property values are read left, top, right and bottom. A Thickness can also be represented with a single value or with two values. If a single value is supplied, all four sides are equal. If two values are supplied they are read (left and right), (top and bottom).*

4. Run your application by pressing F5.
    1. After the application starts, resize the window. Notice the button does not move as the window is resized.
5. Take a minute and change the HorizontalAlignment property values using the Properties Window and notice the different adorners that are rendered. The green arrow in Figure 7 below indicates the HorizontalAlignment adorner.
    1. After changing the HorizontalAlignment property, re-run the application and resize the window. Notice when the button moves or is resized based on how it is aligned as the window is resized.
    2. You can also change the HorizontalAlignment property by clicking on the adorner arrow or the round adorner pointed to by the green arrow.
6. Repeat for the VerticalAlignment property. The blue arrow in Figure 7 below indicates the VerticalAlignment adorner.
7. Return the button back to Top, Left alignment.
8. The red arrow in Figure 7 points to the resize adorner. Clicking and dragging a resize adorner will resize the control in the direction of the mouse movement.

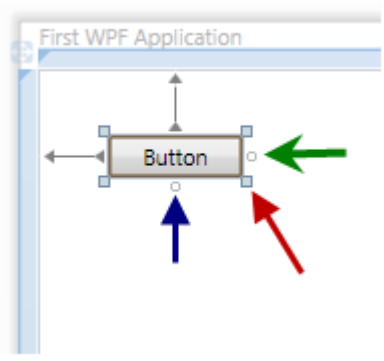

Figure 7 Alignment and Resize Adorners

2. Edge Snaplines
    1. To move a control on the design surface, select it, hold the left mouse down and move the control using the mouse. Release left mouse button when move is complete.
    2. When moving a control near the edge of a container control like a grid an edge snapline will appear. This snapline makes it very easy to uniformly position one or more controls against the containers edge.
    3. In Figure 8 below, notice the two numbers in the left red rectangle. 12 and 12 indicate that the top left corner of the button is 12 pixels from the top and 12 pixels from the left of the grid.
    4. The right red rectangle in Figure 8 contains 191. This indicates the number of pixels from the right edge of the button to the edge of the grid.
    5. When moving a control, if you move the control past the edge snapline, the control will snap to the edge of the container control.

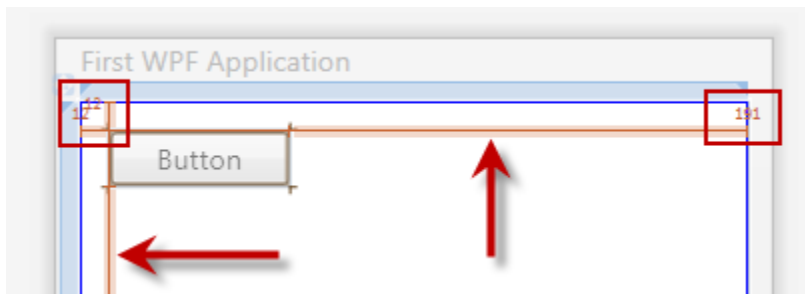

Figure 8 Edge Snaplines

<u>Tip:</u> To turn off Snaplines when moving a control, hold the ALT key down while dragging the control around on the design surface. Use arrow keys to nudge a control, pixel by pixel into perfect position!

3. Control Snaplines
    1. Add another button to the form as picture below.
    2. Figure 9 shows the control snaplines that are displayed when the edges of one control line up with another. The number 20 is the number of pixels that separate the two buttons.

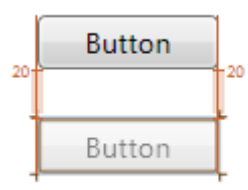

Figure 9 Control Snaplines

4. Text Baseline Snaplines
    1. Figure 10 shows the text baseline snapline that is displayed when two text controls are aligned. The number 20 is the number of pixels that separate the two buttons.

Figure 10 Text Baseline Snaplines

Survey Application – Form Title

At this time delete any buttons or other controls you have placed on the design surface by selecting the control and pressing the DELETE key to delete the control.

Please save your application by pressing CTRL+S or by clicking the Save icon on the ToolBar.

First WPF Application

The rest of this walk through will focus on creating this Simple Survey application. You will learn many new features of the WPF Designer while building this application.

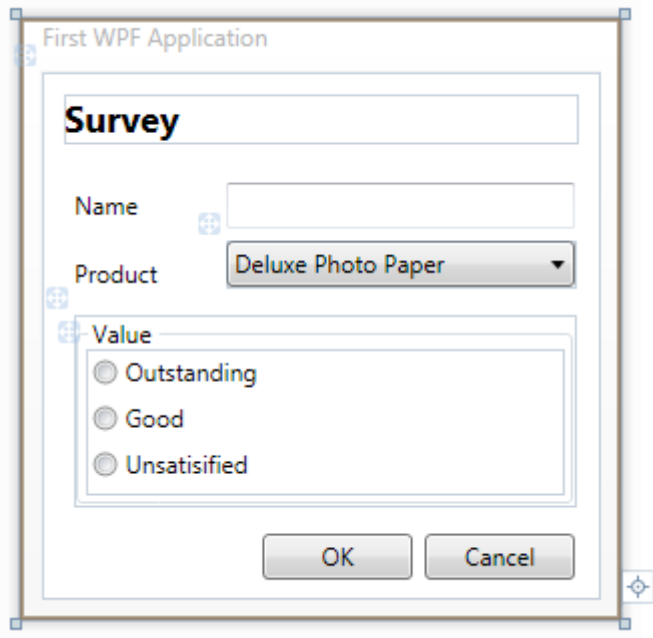

Figure 11 Completed Application

1. Create a TextBlock control to the design surface.
    1. The TextBlock control is located in the Controls tab of the ToolBox.
2. With the new TextBlock selected, set the Properties Window to Alpha View and enter, “text” in the Search Box. Notice as you type, the list of properties is quickly filtered.

    <u>Tip:</u> The search feature of the Properties Window filters the properties and can save you lots of time during application development. Very handy.

3. Change the Text property to Survey.

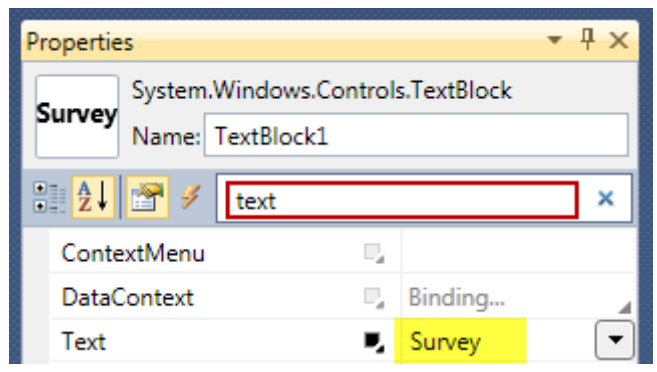

Figure 12 Form Title - TextBlock Text

4. We will now use the Properties Window to position the Form Title TextBlock.
    1. Change the Properties Window to Category View and scroll to the Layout category as shown in Figure 13 below.
    2. Set the HorizontalAlignment, VerticalAlignment and Margin properties as indicated below. Notice how the TextBlock is positioned on the design surface.
5. Using the Search Box, enter font. Use the Font Category editor and increase the Font Size to 18.

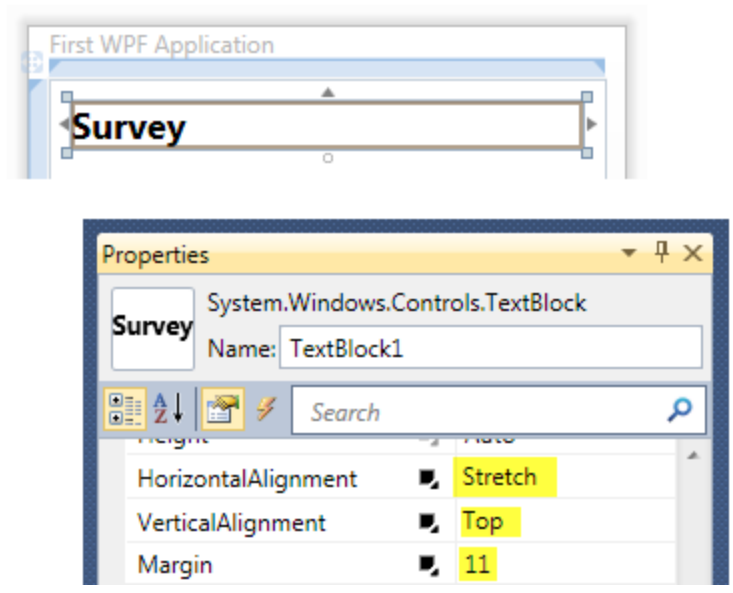

Figure 13 Form Title - TextBlock positioning

<u>Note:</u> In Figures 12 and 13, notice the preview of the TextBlock in the Preview Icon displayed to the left of the control name. This same Preview Icon is also displayed in the Document Outline when the mouse is hovered over controls listed there.

Survey Application – Form Controls Part I

It’s time to give our user interface (UI) some controls the user can interact with at run-time.

1. Add the four controls pictured in Figure 14, two Labels, TextBox and a ComboBox.
    1. Align all controls Top, Left.
    2. Use the control snaplines to assist in aligning and positioning the controls
    3. Select the TextBox. Using the Properties Window change the Name to txtName.
    4. Select the ComboBox. Change the Name to cboProduct.

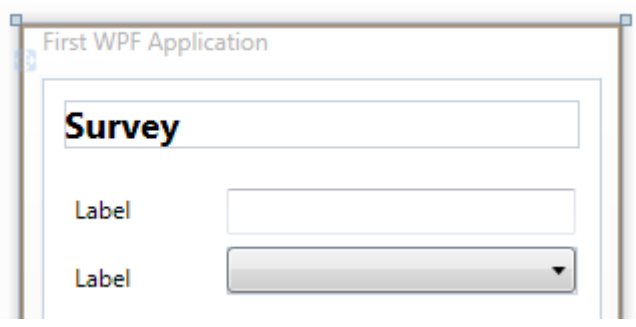

Figure 14 Form Controls

2. Select the top label on the design surface.
    1. Use Properties Window search feature and type in, “con”.
    2. Change the Content property to, “_Name”.
3. Click the bottom label on the design surface.
    1. Notice that the Content property for that label is now selected in the Properties Window.
    2. Change the Content property to, “_Product”.

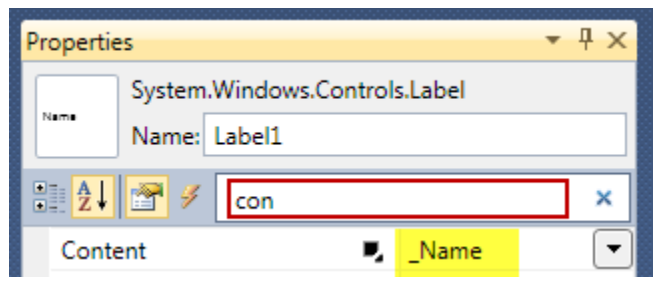

Figure 15 Label Content Property

Note:

The Label control uses the Content property to display its text. You may be thinking why is the text for the Label in the Content property and the text for the TextBlock in the Text property? This is because the two properties are different data types with different associated features.

The Label Content property can contain just about anything, text, image, grid, etc.

The Label control can associate an Access Key with another control, so that when the Access Key is pressed, focus will be moved to the target control. The Label Content property plays a role in this.

The TextBlock Text property is special. It can contain text or inline content flow elements. See http://msdn.microsoft.com/en-us/library/bb613554.aspx for more information on inline content. Inline content allows you to have multi-line formatted content in a TextBlock.

4. Implementing a label Access Key requires an Access Key and Target control assignment.
    1. "_N" associates the key combination ALT+N with this label's Access Key. When the user holds down the ALT key the label will be displayed as "Name".
    2. In WPF the Target control is assigned using data binding.
    3. The Properties Window Data Binding Builder provides the UI for creating the required data binding.
5. Setting the Target Property for the Access Key.
    1. In the following steps we'll create what is called an "element name binding." Element name bindings enabled a control to bind to property on another named object.
    2. Select the TextBox control and enter "tar" in the Properties Window Search Box. See Figure 16 below.
    3. The red arrow is pointing to the Property Marker. The Property Marker icon changes based on property value assignment and where the property is being assigned.
    4. Click the Property Marker and the context menu is displayed.
    5. Selected Apply Data Binding…

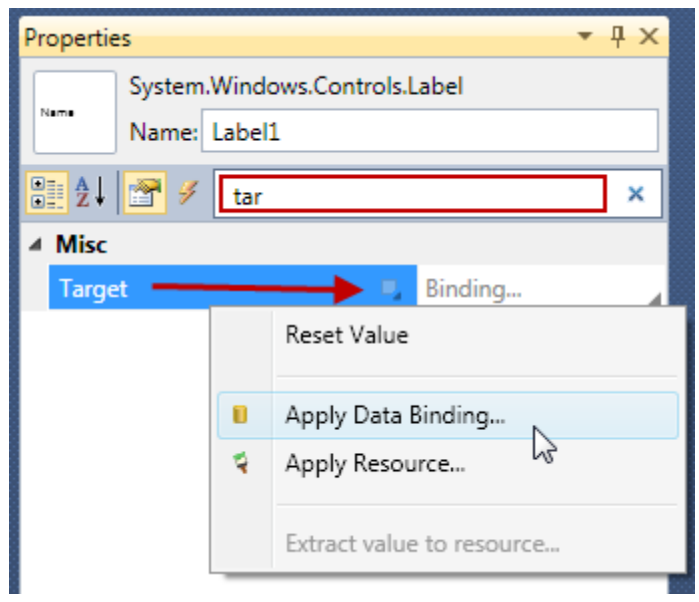

Figure 16 Binding Target Property

    6. In the Source tab, select ElementName. A list of named UI controls will be displayed.
    7. Select txtName. Notice the Source tab title is updated as you perform the selections.
        1. The Target binding does not require that a Path be assigned.
        2. To close the Data Binding Building press ENTER or click outside the Data Binding Builder.

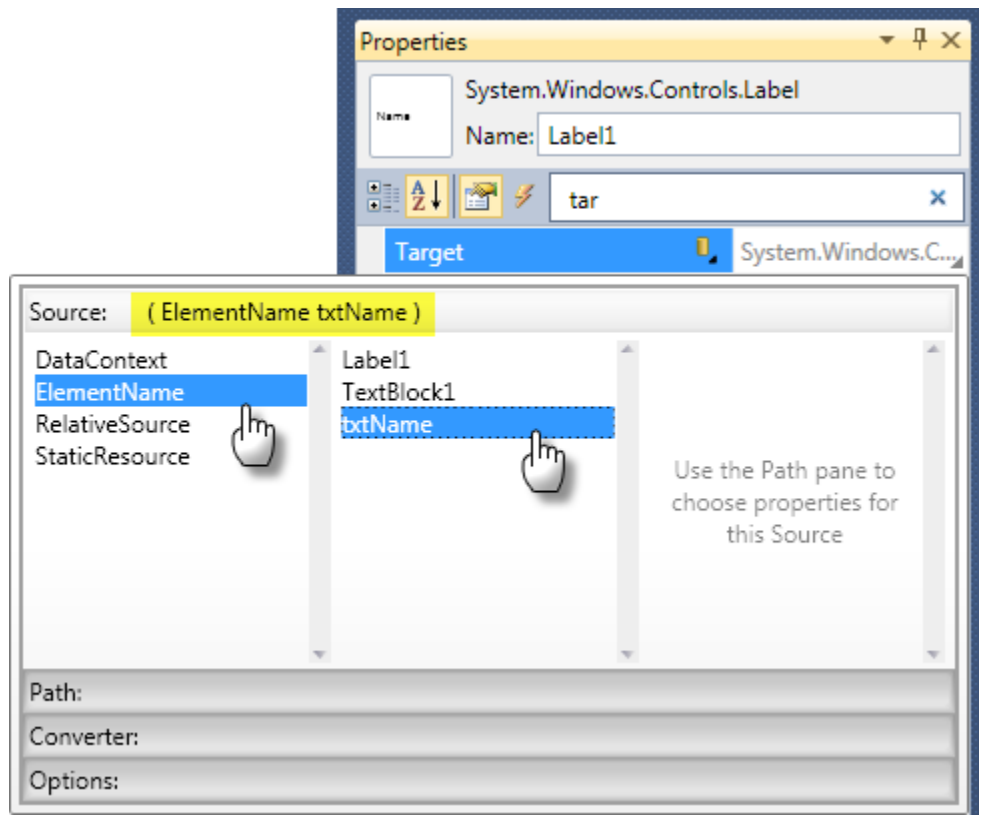

Figure 17 Data Binding Builder

    8. Let's test our ALT+N Access Key. Run your application and press ALT+N. Focus is moved to the TextBox control.
6. Repeat the above steps for the Product label, associating ALT+P with the ComboBox as the Target.

Important:

You have just data bound labels to their associated TextBox to enable Access Key control selection. Data binding in WPF is one of the most powerful features of the platform and is used extensively in all WPF applications. For example, binding UI controls to database or collection data, or intra-form binding that provides an interactive experience for users of your application. You will find that data binding in the UI will save you from writing a lot of code you would normally write if developing on another platform.

A proper understand of WPF Data Binding is essential for the WPF developer. MSDN documentation provides clear and concise information that you can read here http://msdn.microsoft.com/en-us/library/ms752347.aspx.

7. There are several ways to add items to a ComboBox. In this walk through we will add three static items to the ComboBox.
    1. Select the ComboBox.
    2. Search for items in the Properties Window. See Figure 18 below.
    3. Click the Items property ellipsis icon to open the Collection Editor for the Items property.

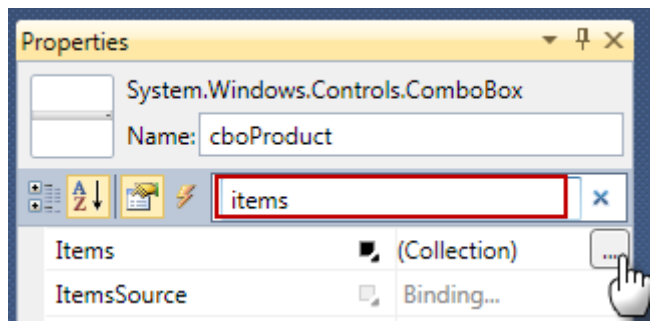

Figure 18 ComboBox Items Collection

8. Adding items is very easy using the Collection Editor.
    1. Click on the Add button.
    2. Locate the Content property and change the value.
    3. You can also order the items or remove an item.
9. Using the Collection Editor in Figure 19, add three items, setting the Content property to the following strings:
    1. Deluxe Photo Paper
    2. Financial Calculator
    3. 4GB USB Thumb Drive
10. Click OK when done.
    1. In the XAML Editor, look at the XAML markup that was generated for you.

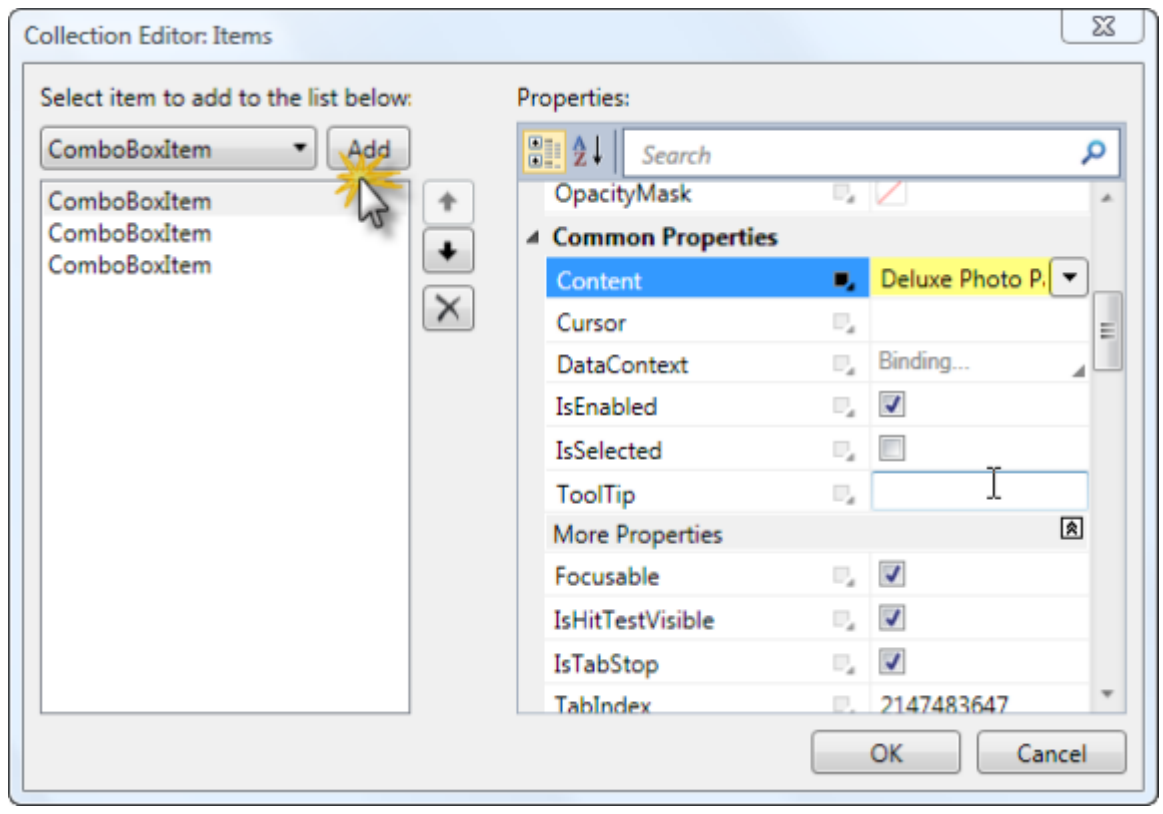

Figure 19 Collection Editor

Survey Application – Form Controls Part II

In this section we will use a GroupBox to display RadioButtons. RadioButtons are mutually exclusive; meaning that when one is selected the previously selected RadioButton is deselected automatically.

We will also introduce a new panel control, the StackPanel. StackPanels stack their child controls either horizontally or vertically based on their Orientation property.

1. Below the ComboBox add a GroupBox control from the ToolBox Controls tab.
2. The GroupBox control can contain a single child control. The WPF Designer automatically adds a child Grid control for you. See Figure 20 below.
3. We want to replace the Grid with a StackPanel. You can either edit the XAML markup directly by changing to or you can use the designer.
4. Select the GroupBox Grid by clicking inside the GroupBox.
    1. Press the DELETE key to delete the child Grid.

Figure 20 GroupBox

5. To add a StackPanel as a child of the GroupBox drag the StackPanel control from the ToolBox to the GroupBox. See Figure 21 below. Notice the blue adorner surrounding the Border. This indicates the parent control that your StackPanel will be a child of if dropped at the current location.

Figure 21 Parenting Control on Creation

6. When controls are added to the design surface using the ToolBox some properties are given default values.

   Tip: You can also drag controls from the ToolBox to the XAML Editor. When a control is dragged to the XAML editor, no properties are given default values.

   1. After creating the StackPanel the design surface will look similar to Figure 22 below. Notice in the XAML Editor and Document Outline that the StackPanel is a child of the GroupBox.

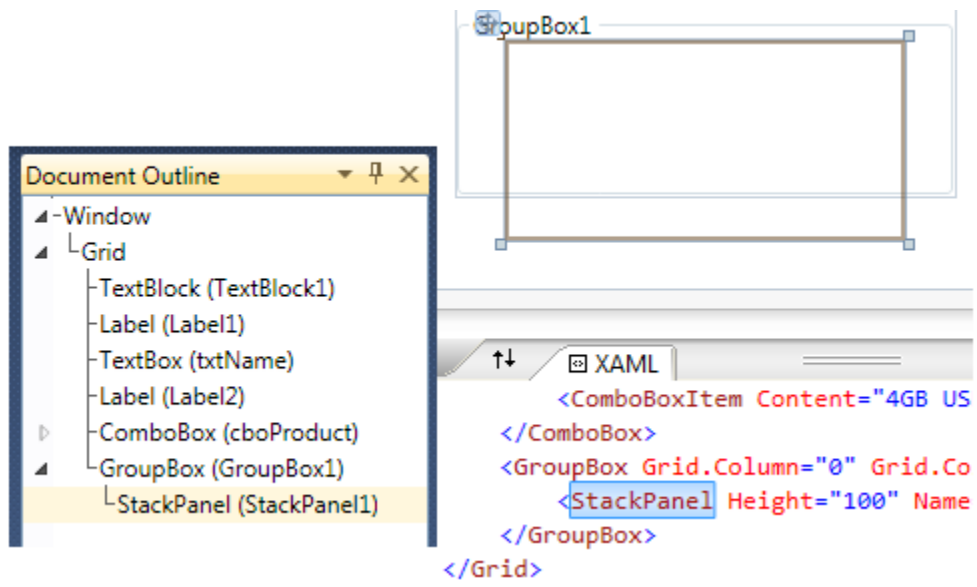

Figure 22 StackPanel as a child of the GroupBox

7. For our application we want the StackPanel to automatically consume all the space inside the GroupBox so that if the GroupBox is resized at design or run-time, the StackPanel will grow or shrink with the GroupBox.
8. Let's use the Properties Window to accomplish this task. See Figure 23 below.
   1. With the StackPanel selected, search for "wid" in the Properties Window.
   2. Click the Property Marker for the Width property and select Reset Value. This will change the Width property to Auto.

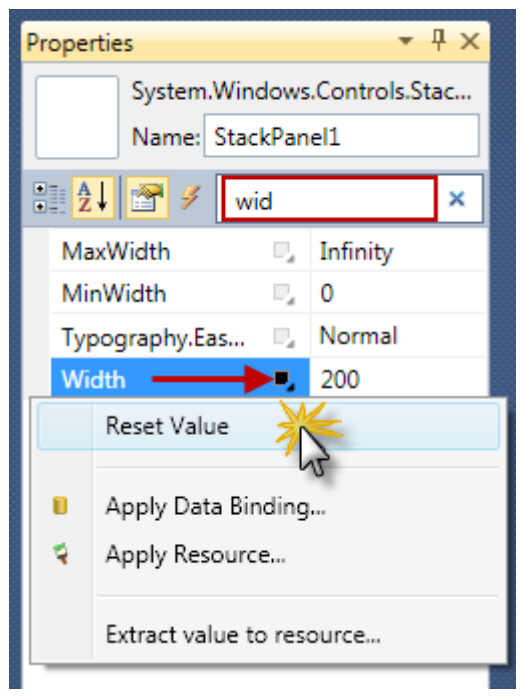

Figure 23 Resetting Property Values

   3. Repeat for the Height property, setting its value to Auto.
   4. Search for "align". Reset the property value for the HorizontalAlignment and VerticalAlignment properties. When reset, their value will be changed to Stretch. The StackPanel is now auto sized and should look like Figure 24.

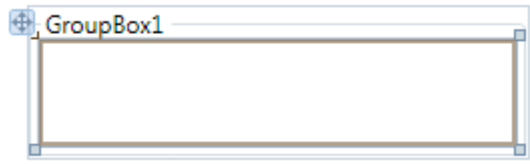

Figure 24 Auto Sized StackPanel

9. Let's give the GroupBox a meaningful heading. Using the Properties Window search for "head" and change the Header property to "Value". See Figure 25 below.
10. With the StackPanel selected, add three RadioButtons as children of the StackPanel by double clicking the RadioButton in the ToolBox three times. When completed the design surface will look similar to Figure 25 below.
11. A powerful feature of the WPF Designer is the ability to select multiple controls and edit common property values.
12. Select the three RadioButtons by control clicking them. (CTRL+Click). Notice that the Properties Window name changes to, "Multiple objects selected."
    1. Use the Properties Window to reset the Height and Width property values for the selected RadioButtons.
    2. Using the Properties Window, search for "margin" and set the value to 3.5. Depending on the size of your GroupBox the setting of the Margin property may cause one or more RadioButtons to disappear from view, we will correct that next.

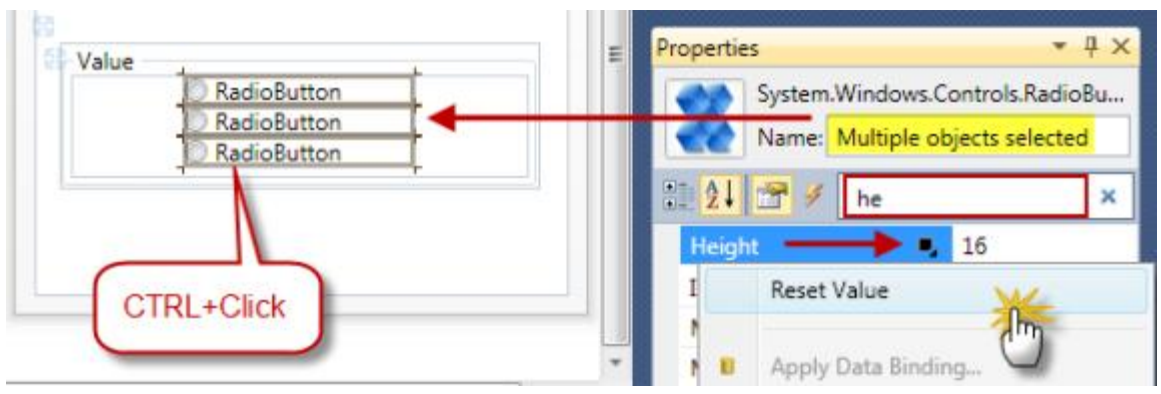

Figure 25 Multi-Selection Control Editing

Tip: *You can also multi-select all controls in a container by selecting the container and pressing CTRL+A. In figure 25 above, select the StackPanel and press CTRL+A to select the three RadioButtons.*

13. On the design surface select the GroupBox, use the resize adorner on the bottom and resize the GroupBox so it looks like Figure 26 below.
    1. Notice how the child StackPanel automatically resizes when its parent GroupBox is resized.

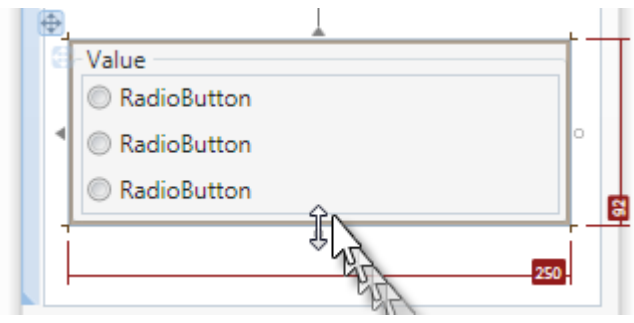

Figure 26 Resizing GroupBox

14. Use the Properties Window to assign the following values to the three RadioButtons Content properties.
    1. Outstanding
    2. Good
    3. Unsatisfied

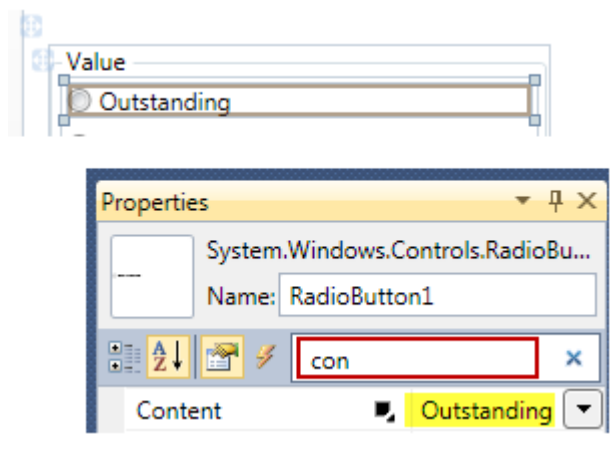

Figure 27 RadioButton Content

15. You can set a default value for the ComboBox by assigning a value to the SelectedIndex property.
16. Select the ComboBox, use the Properties Window change the SelectedIndex property to 0.
    1. Notice how the ComboBox now displays the value of the ComboBoxItem at index 0. See Figure 28 below.

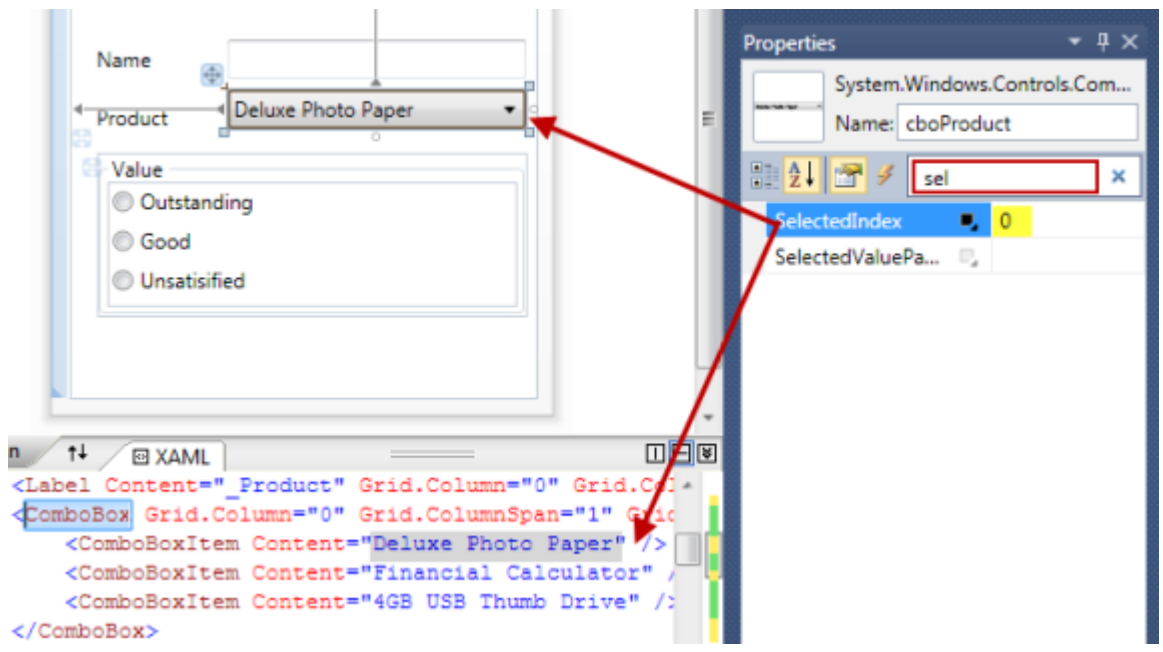

Figure 28 ComboBox Selected Index

Survey Application – Form Controls Part III

To complete our application we need to add two Buttons and add some code that will execute when the Button is clicked.

1. Add two buttons as pictured in Figure 29 below.
    1. Change the left Button Content property to OK and set the Name property to btnOK.
    2. Change the right Button Content property to Cancel and set the Name property to btnCancel.
2. The WPF Designer provides three ways to wire up code to a control event.
    1. Double clicking a control will wire up the default event.
    2. Using the Properties Window Events tab, locate the desired event and double click the region indicated in Figure 29.
        1. If the event code is already in the code behind, you can select the method name in the ComboBox instead of double clicking to create a new event handler.
    3. Using the XAML Editor to create a new event handler or select a method.
3. Select the OK Button. Using the Properties Window Events tab, double click next to the Click event. This will wire up the Click event handler and add code the code behind.

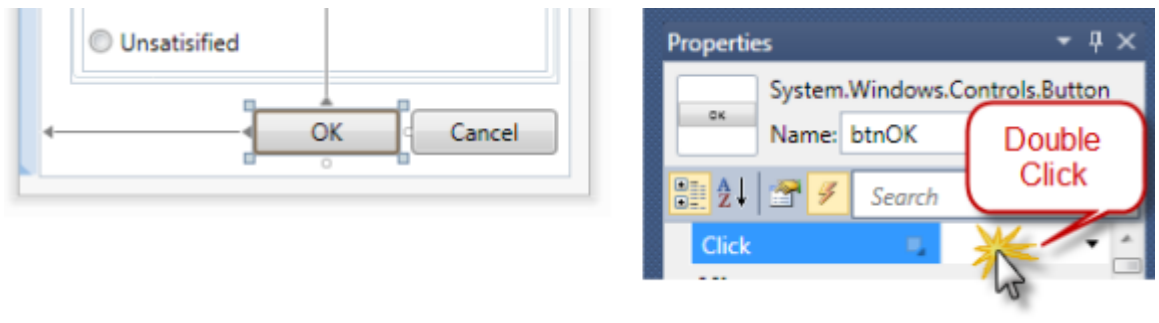

Figure 29 Properties Window Events Tab

    1. Visual Basic wires up the event using the Handles construct.
    2. C# wires up the event by adding the event name and event handler to the XAML markup.
4. On the design surface, double click the Cancel Button to generate the event handler for this Button.
    1. Add the MessageBox.Show and Close method calls to the generated event handlers.

Visual Basic Code

```
Class Window1

    Private Sub btnCancel_Click(
            ByVal sender As System.Object,
```

```
            ByVal e As System.Windows.RoutedEventArgs
            ) Handles btnCancel.Click

        Me.Close()
    End Sub

    Private Sub btnOK_Click(
            ByVal sender As System.Object,
            ByVal e As System.Windows.RoutedEventArgs
            ) Handles btnOK.Click

        MessageBox.Show("Thank you for your feedback")
        Me.Close()
    End Sub

End Class
```

C# Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace FirstWPFApplication
{
    ///
    /// Interaction logic for Window1.xaml
    ///
    public partial class Window1 : Window {
        public Window1() {
            InitializeComponent();
        }

        private void btnOK_Click(object sender, RoutedEventArgs e) {
            MessageBox.Show("Thank you for your feedback.");
            this.Close();
   }

        private void btnCancel_Click(object sender, RoutedEventArgs e) {
            this.Close();

        }
    }
}
```

5. Run your completed application by pressing F5.
    1. Use the Access Key ALT+N to set focus to the TextBox.
    2. Now use ALT+P to set focus to the ComboBox.
        1. Use arrow keys to change the ComboBox value.
        2. Press F4 to open the ComboBox dropdown.
    3. Click the RadioButtons to see how only one is selected at time.
    4. Clicking Cancel will close the application.
    5. Clicking OK will cause the MessageBox to display its message. Close the MessageBox and the application will close.

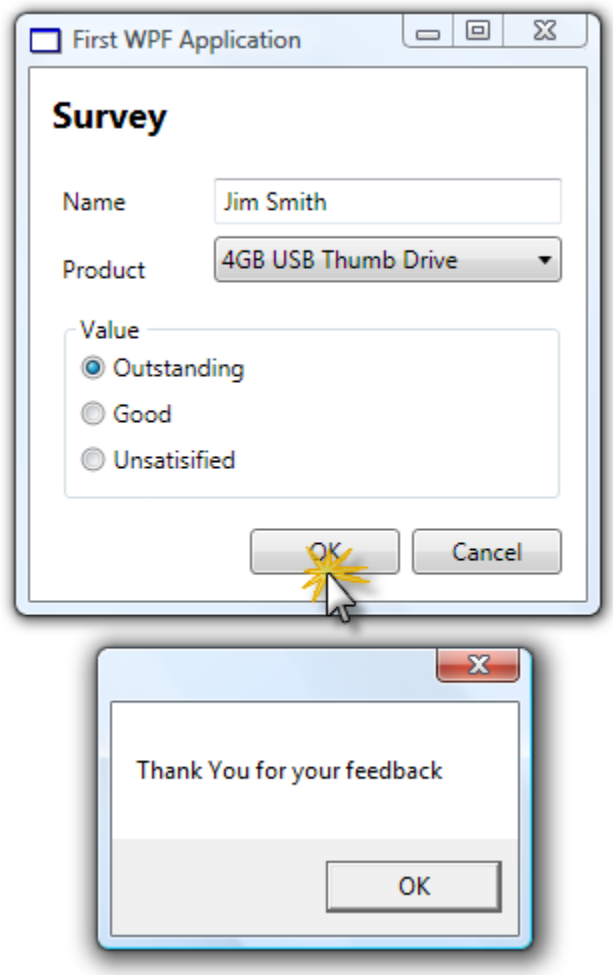

Figure 30 Running the Application