course: COMP6210 week: 03 lesson: 01 topic: databinding

DOWNLOAD PDF CLICK HERE

Data Binding

When using XAML we can use databinding to connect the UI of one object to the datasource of another.

This saves us from writing the C# code that we would normally need to write.

We do this by using the `Binding` Expression inside of the value of the element that we want to get the data into like so:

```
Binding ElementName=name Path=Value
```

For example inside of a label, this would look like:

```
<Label Content="{Binding ElementName=name Path=Value}" />
```

or like this:

```
<Label Content="{Binding Source={x:Reference name}, Path=Value}" />
```

We can also specify how the data is formatted, but that is slightly different for each object based on its type.

Bind data to a parent objects

We are able to bind the same data to multiple parts of an UI object. We do this by using the `DataContext` which links the UI object and then we can access the properties in the parameters.

Unlike the above code where we can access the ElementName attribute, we must use the `x:Reference` accessor to link to the UI object.

```
<Label ....// other attributes
DataContext="{x:Reference mySlider}"
Opacity="{Binding Path=Value}"
Content="{Binding Path=Value}" ContentStringFormat="{}{0:F2}" />
```

We can do something similar with parent objects like this:

```
<StackPanel Height="210" Orientation="Horizontal" DataContext="{x:Reference mySlider}">

  <Label Width="200" FontSize="30" VerticalAlignment="Center" HorizontalContentAlignment="Center" Margin="50, 0, 0, 0" Opacity="{Binding I

  <Rectangle Fill="Blue" Width="100" Height="100" Opacity="{Binding Path=Value}" />

</StackPanel>
```

In the above code - all of the bindings are from the source that is defined in the StackPanel.

Binding a Collection to XAML

It is very easy to use code to link a datasource to a `ListBox` like this"

```
//XAML
<ListBox Name="FruitBox" Width="200" Height="400" />

//C#
string[] fruits = new string[3]("Apple", "Banana", "Orange");
FruitBox.ItemsSource = fruits;
```

But what about if our collection is a little more complex, like an object with multiple properties?

Let's say that our code is something like this:

```
public sealed class ItemInStock
{
    public string Name { get; set; }
    public int Quantity { get; set; }
}

public partial class MainWindow : Window
{
  List<ItemInStock> ItemsInStock = new List<ItemInStock>()
  {
      new ItemInStock { Name = "Apple", Quantity = 5 },
      new ItemInStock { Name = "Banana", Quantity = 5 },
      new ItemInStock { Name = "Mandarin", Quantity = 5 },
      new ItemInStock { Name = "Orange", Quantity = 5 },
      new ItemInStock { Name = "Peer", Quantity = 5 }
  };

  public MainWindow()
  {
      InitializeComponent();

      StockList.ItemsSource = ItemsInStock;
  }
}
```

Inside our XAML code we can create a template that allows us to display this data.

Let's have a look at the template:

```
<ListView Name="StockList" ItemsSource="{Binding Contents}" Height="170" Margin="20,20,20,0">

    <ListView.View>

        <GridView >
            <GridView.Columns>
                <GridViewColumn Header="Name" Width="auto" >
                    <GridViewColumn.CellTemplate>
                        <DataTemplate>
                            <StackPanel>
                                <Label Content="{Binding Path=Name}"/>
                            </StackPanel>
                        </DataTemplate>
                    </GridViewColumn.CellTemplate>
                </GridViewColumn>
                <GridViewColumn Header="Quantity" Width="auto"  >
                    <GridViewColumn.CellTemplate>
                        <DataTemplate>
                            <Label Content="{Binding Path=Quantity}" />
                        </DataTemplate>
                    </GridViewColumn.CellTemplate>
                </GridViewColumn>
            </GridView.Columns>
        </GridView>

    </ListView.View>

</ListView>
```

We can use a `ListView` to display our list and use a `GridView` to display multiple columns, if we need to.

As the hierarchy goes deeper, you can see we are just using `Template` type tags. In addition to the template type tags, the stuff we actually see (labels and the like) are just used as per normal.

You can read up more about how the structure works here.