

UWP-040 - Data Binding to the GridView and ListView Controls

In this lesson we will talk about data binding, and a couple of Controls that will allow us to bind data in a grid/list type fashion. Data binding has been around for a long time, and basically it's the process of taking data from a source, and then associating it with elements of a user interface.

Consider having an object in your code – one that describes a book, or a car, etc – and you then want to be able to take that data and display it onto a user interface. That is essentially what data binding lets you do. It lets you specify a template, of sorts, that each object instance uses in order to be represented within the user interface.

Let's start by looking at a basic example project, for demonstration purposes. Here is a simple grid-like layout which we can use as our starting point

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d">

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}" Margin="20">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="100" />
    </Grid.RowDefinitions>

    <TextBlock Grid.Row="1"
        Name="ResultTextBlock"
        FontSize="24"
        Foreground="Red"
        FontWeight="Bold"
        Margin="0,20,0,0" />

</Grid>
</Page>
```

Now, let's refer to some of our basic class information, which will just hold basic data that we can populate later

```
namespace xBindDataExample.Models
{
    public class Book
    {
        public int BookId { get; set; }
        public string Title { get; set; }
        public string Author { get; set; }
        public string CoverImage { get; set; }
    }
}
```

And, next we have a BookManager class that is responsible for populating instances of each Book, and then adding them to a List collection

```
public class BookManager
{
    public static List<Book> GetBooks()
    {
        var books = new List<Book>();

        books.Add(new Book { BookId = 1, Title = "Vulpate", Author = "Futurum", CoverImage="Assets/1.png" });
        books.Add(new Book { BookId = 2, Title = "Mazim", Author = "Sequiter Que", CoverImage = "Assets/2.png" });
        books.Add(new Book { BookId = 3, Title = "Elit", Author = "Tempor", CoverImage = "Assets/3.png" });
    }
}
```

Now, in MainPage.xaml, one of the things you will need to do is establish a GridView template, describing how each individual instance of Book will appear on-screen.

```
<Grid Background="{ThemeResource ApplicationPageBack
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="100" />
    </Grid.RowDefinitions>

    <GridView>
        <GridView.ItemTemplate>
            <DataTemplate>
                <Image Width="150" />
                <TextBlock FontSize="16" />
                <TextBlock FontSize="10" />
            </DataTemplate>
        </GridView.ItemTemplate>
    </GridView>

    <TextBlock Grid.Row="1"
```

The next thing is to tell the GridView what it should, ultimately, bind to. In order to do this, set the ItemSource property to what you want it to bind to

```
<GridView ItemsSource="{x:Bind Books}">
  <GridView.ItemTemplate>
    <DataTemplate>
      <Image Width="150" />
    </DataTemplate>
  </GridView.ItemTemplate>
</GridView>
```





This, in turn, will refer to a public property called Books in MainPage.xaml. This property will be of type List<Book>, and will store the list of Books returned from the BookManager.GetBooks()

But before that, we have to set a property on the DataTemplate called "x:DataType" and set it to type Book. And, in order to reference this class type in my XAML, you will need to add a XML namespace up at the page level.

```
xmlns:data="using:xBindDataExample.Models"
mc:Ignorable="d">

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="100" />
  </Grid.RowDefinitions>

  <GridView ItemsSource="{x:Bind Books}">
    <GridView.ItemTemplate>
      <DataTemplate x:DataType="Book">
        <Image Width="150" />
      </DataTemplate>
    </GridView.ItemTemplate>
  </GridView>
</Grid>
```



This namespace reference just refers to where the Book class is located. In this case, it's in the Models namespace, as it is in the "Models" folder within the project.



With that set up, you should now be able to properly reference that class by changing the x:DataType as follows

```
<DataTemplate x:DataType="data:Book">
```

Now, we need to set properties for each Control, and have them refer to each property of the Book class.

```
<GridView ItemsSource="{x:Bind Books}">
  <GridView.ItemTemplate>
    <DataTemplate x:DataType="data:Book">
      <StackPanel>
        <Image Width="150" Source="{x:Bind CoverImage}" />
        <TextBlock FontSize="16" Text="{x:Bind Title}" />
        <TextBlock FontSize="10" Text="{x:Bind Author}" />
      </StackPanel>
    </DataTemplate>
  </GridView.ItemTemplate>
</GridView>
```



With that out of the way, all that's left to do is to populate that property called "Books" – that we're ultimately binding to - within MainPage.xaml.cs

```
<GridView ItemsSource="{x:Bind Books}">
  <GridView.ItemTemplate>
    <DataTemplate>
      <Image Width="150" />
    </DataTemplate>
  </GridView.ItemTemplate>
</GridView>
```



In MainPage.xaml.cs make sure you have the necessary using statement relative to your project, to make the binding happen

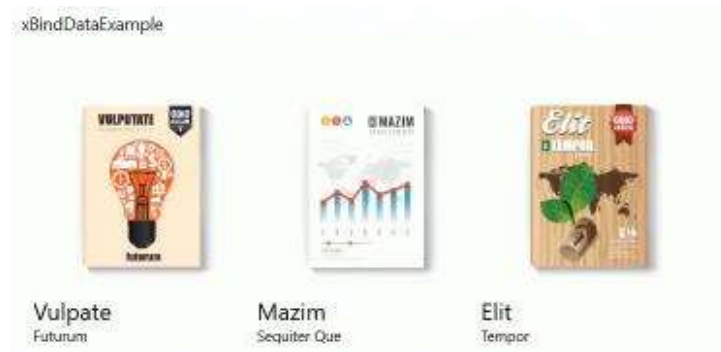
```
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using xBindDataExample.Models;
```

And then populate those instances

```
public sealed partial class MainPage : Page
{
    private List<Book> Books;

    public MainPage()
    {
        this.InitializeComponent();
        Books = BookManager.GetBooks();
    }
}
```

And when the program runs, the XML will now magically bind to that data set



To pretty this up a bit, you can center the elements back in the XML

```
<StackPanel HorizontalAlignment="Center">  
    <Image Width="150" Source="{x:Bind CoverImage}" />  
    <TextBlock FontSize="16" Text="{x:Bind Title}" HorizontalAlignment="Center" />  
</StackPanel>
```

Now, to retrieve information when selecting on each item let's set some attributes to the GridView. First, make it clickable.

```
<GridView ItemsSource="{x:Bind Books}" IsItemClickEnabled="True">
```

And then assign it to an event, add the following

```
IsItemClickEnabled="True" ItemClick="GridView_ItemClick">
```

Which you can edit in the MainPage.xaml.cs

```
private void GridView_ItemClick(object sender, ItemClickEventArgs e)  
{  
    var book = (Book)e.ClickedItem;  
}
```

The above code gives you the instance of the item that was clicked on. And then outputs that book title to the screen (as shown below)

```
private void GridView_ItemClick(object sender, ItemClickEventArgs e)  
{  
    var book = (Book)e.ClickedItem;  
    ResultTextBlock.Text = "You selected " + book.Title;  
}
```




Now, if you want to reuse the styling information as a template (especially for multiple app sections that refer to it), you can take it out of the GridView.ItemTemplate (cut)

```
<GridView.ItemTemplate>
  <DataTemplate x:DataType="data:Book">
    <StackPanel HorizontalAlignment="Center">
      <Image Width="150" Source="{x:Bind CoverImage}" />
      <TextBlock FontSize="16" Text="{x:Bind Title}" Horizontal
      <TextBlock FontSize="10" Text="{x:Bind Author}" Horizontal
    </StackPanel>
  </DataTemplate>
```

And insert it into a page resource at the top of your document (paste)

```
mc:Ignorable="d">
<Page.Resources>
  <DataTemplate x:DataType="data:Book">
    <StackPanel HorizontalAlignment="Center">
      <Image Width="150" Source="{x:Bind CoverImage}" />
      <TextBlock FontSize="16" Text="{x:Bind Title}" Horizontal
      <TextBlock FontSize="10" Text="{x:Bind Author}" Horizontal
    </StackPanel>
  </DataTemplate>
</Page.Resources>
```

And then give that an x:Key, so you can refer to it in the rest of the document

```
<DataTemplate x:DataType="data:Book" x:Key="BookDataTemplate">
```

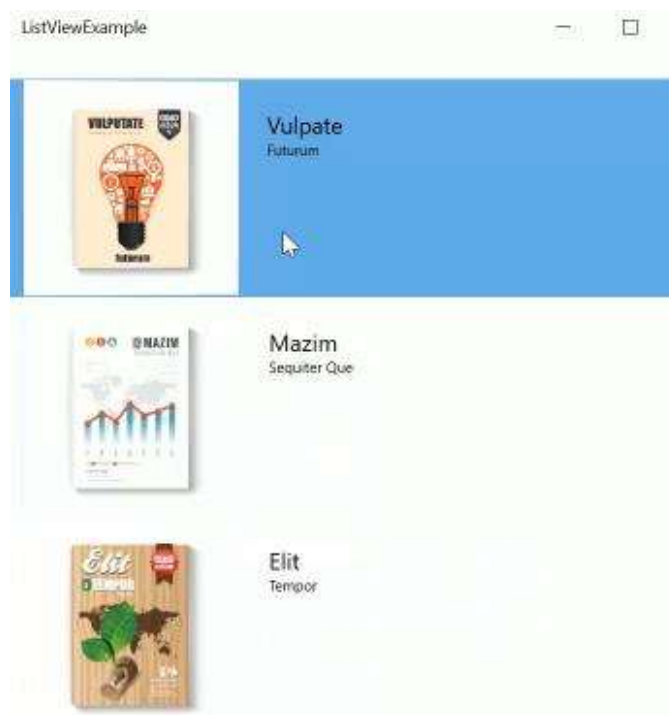
And reference it within the GridView as follows

```
<GridView ItemsSource="{x:Bind Books}"
  IsItemClickEnabled="True"
  ItemClick="GridView_ItemClick"
  ItemTemplate="{StaticResource BookDataTemplate}">
</GridView>
```

One other quick thing to demonstrate is how you can take this basic styling structure and, instead of displaying in a grid-like format, have it display in a list-like format (the information becomes stacked, one on top of the other)

```
<DataTemplate x:Key="BookListDataTemplate" x:DataType="data:Book">
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Left">
        <Image Name="image" Source="{x:Bind CoverImage}" HorizontalAlignm
        <StackPanel Margin="20,20,0,0">
            <TextBlock Text="{x:Bind Title}" HorizontalAlignment="Left" F
            <TextBlock Text="{x:Bind Author}" HorizontalAlignment="Left"
        </StackPanel>
    </StackPanel>
</DataTemplate>
```

And with such a simple modification you can now display in a list format that looks something like this



So, now we can take what we learned about the DataTemplate, how to bind to the data, and the difference between the GridView and the ListView, and apply that to the coming lessons.