# ICE DataBinding-ListView

## Part 1 – Data Binding

We are going to expand the databinding work to work on binding to items on a listview. ListViews are popular controls to show lists of information. Data binding can be used to show the data in a listview in an organized way.

For this example we will be making a list of game levels to display so that we can have the use select one to start their game.

1. Create a new blank Xamarin project and name it DatabindingListview. Do NOT put a – (dash) in the project name, this will cause issues with the namespace created.
2. Update the project to the latest Xamarin Form (4.8+).
3. In the DataBindingListview project, create two new folders named **Models** and **ViewModels**. We will use these folders to keep the models and viewmodels organized.
4. In the Models folder, create a new class named GameLevel. Add properties for the Name, LevelNumber, MinPlayerLevel, and MaxPlayerLevel. This will look like this:  
     
    class GameLevel

{

public string Name { get; set; }

public int LevelNumber { get; set; }

public int MinPlayerLevel { get; set; }

public int MaxPlayerLevel { get; set; }

}

1. In the DataModels folder, create a new class named MainPageViewModel. In the viewModel, create a List<GameLevel> as a property. This will be used to display the information in the list view. Fill in a few records in this list in the constructor. The code will look something like this (come up with your own data and not just copying mine):  
     
   class MainPageViewModel

{

public List<GameLevel> Levels { get; set; }

public MainPageViewModel()

{

Levels = new List<GameLevel>();

Levels.Add(new GameLevel { Name="N00b", LevelNumber=0,   
 MinPlayerLevel=1, MaxPlayerLevel=5 });

Levels.Add(new GameLevel { Name="Average", LevelNumber=1,   
 MinPlayerLevel=4, MaxPlayerLevel=15 });

Levels.Add(new GameLevel { Name="Don't Try It", LevelNumber=2,  
 MinPlayerLevel=15, MaxPlayerLevel=50 });

}

}  
  
You can also use a shortcut and define the levels like this:

Levels = new List<GameLevel>

{

new GameLevel { Name="N00b", LevelNumber=0, MinPlayerLevel=1, MaxPlayerLevel=5 },

new GameLevel { Name="Average", LevelNumber=1, MinPlayerLevel=4, MaxPlayerLevel=15 },

new GameLevel { Name="Don't", LevelNumber=2, MinPlayerLevel=15, MaxPlayerLevel=50 }

};

1. In the MainPage.xaml file remove the StackLayout and everything inside of it and replace it with a **ListView**. DataBind the **ItemSource** to **Levels**.

<ListView ItemsSource="{Binding Levels}">

</ListView>

1. At the top on the MainPage.xaml file we need to add a new namespace to be able to get to the view models right from the xaml code. Add this to the ConstentPage node definition:

xmlns:vm="clr-namespace:DataBindingListview.ViewModels"

1. Just under that node and before the ListView add the code to set the BindingContext.

<ContentPage.BindingContext>

<vm:MainPageViewModel />

</ContentPage.BindingContext>

Run the program and you should see something that looks like this:

A screenshot of a cell phone

Description automatically generated

## Part 2 – Cell Layout

At this point we have our data coming from the ViewModel and need to get it formatted to look nice and to show the actual data instead of just the data type.

1. The first thing that we can do is to go to the data model and to add a ToString method so that we can see all of the fields properly. Add the ToString to the GameLevel class and you will see something that looks like this:

A screenshot of a cell phone

Description automatically generated  
  
This is not a bad start, but we can do better with data binding.

1. From here we can create our own look and feel for each cell in the ListView. Inside the code for the Listview we can add a DataTemplate for how the data is suppose to look.

<ListView ItemsSource="{Binding Levels}">

<ListView.ItemTemplate>

<DataTemplate>

<StackLayout Padding="10">

<Label Text="{Binding LevelName}"

FontSize="16" />

<Label Text="{Binding LevelNumber, StringFormat='Level: {0}'}"

FontSize="13" />

</StackLayout>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

When you run this now you will get a runtime error. This is because the ListView is for showing things like **TextCell** objects and not a custom look like we defined. So we need to change the ListView in the xaml to be a Collectionview instead. This was introduced around Xamarin Forms 4.3 and gives the user more control on the layout. Change the ListView and ListView.ItemTemplate to be CollectionView and CollectionView.ItemTemplate.

Now the code should run and your output will look like this:

A screenshot of a cell phone

Description automatically generated

## Part 3 – Expanding GameLevel

From here, the app works pretty well. Add some extra fields to the data model and get that information to print out. You can do things like getting more than one thing to print on a single line using a horizontaol StackLayout, add images (play with that f you want), etc. Add at least 2 more fields and get them to be displayed.

## Part 4 – Adding Levels

At this point, we have a ViewModel that is holding our data and we can display it in the **CollectionView**. Let’s add a button to the main page to be able to add more Levels.

1. Go to the MainPage and add a StackLayout around the CollectionView. Above the CollectionView, add a Button and give it a Text of “Add Level” and add a Command and data bind t to a command called AddCommand.
2. Go to the ViewModel and add a new property of Command type and call it AddCommand.
3. Create a new method in the ViewModel that will be private named Add to hold the logic of adding a new GameLevel.

private void Add()

{

Levels.Add(new GameLevel { LevelNumber = Levels.Count,   
 LevelName = "added level " + Levels.Count });

}

1. In the constructor, add a line at the end to create the command.  
     
   AddCommand = new Command(Add);

From here you can add a breakpoint in the Add method to make sure that we are adding new levels to the list of levels. Of course you will notice that no new records are being shown in the CollectionView.

1. To get the data to update we need to change the List<GameLevel> to be an ObservableCollection<GameLevel>. ObservableCollection will tell the GUI that it is updated when you add or remove an item.  
     
   when you run now, you should see it adding levels to the list for every button click.
2. Add another button to the GUI to remove the last one from the list.

**Submission: ZIP and Post to the dropbox**