# ICE Paging

## Part 1 – Create Multiple Pages

We are going to expand our sample applications and make them use multiple pages. The first part will be to setup the pages, the data model, and the ViewModel that we will be using. The first page will be a CollectionView of items and when we click on an item, we will open a second page to view the data.

For this example we will be making a list of Video Games that we will display in the list. Then when you select one of them, it will openup a page to show all of the details of that game that we have saved.

1. Create a new blank Xamarin project and name it Paging.
2. Update the project to the latest Xamarin Form (4.8+).
3. In the Paging project, create two new folders named **Models** and **ViewModels**. We will use these folders to keep the models and viewmodels organized.

In the Models folder, create a new class named GameData. Add properties for the Name, YearOfRelease, Rating, and Platform. Platform will use an enum of the various platforms. This will look like this:  
  
 enum PlatformType

{

Arcade,

Board,

PC,

Console,

Handheld,

Mobile

}

class GameData

{

public string Name { get; set; }

public int YearOfRelease { get; set; }

public float Rating { get; set; }

public PlatformType Platform { get; set; }

}

1. In the ViewModels folder, create two new classes named MainPageViewModel and DetailPageViewModel. In the MainPageViewModel, create a List<GameData> 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<GameData> Games { get; set; }

public MainPageViewModel()

{

Games = new List<GameData>

{

new GameData { Name="Qbert", YearOfRelease=1982, Rating=3,  
 Platform=PlatformType.Arcade },

new GameData { Name="Starcraft", YearOfRelease=1998, Rating=5,  
 Platform=PlatformType.PC },

new GameData { Name="Doom", YearOfRelease=1993, Rating=4,  
 Platform=PlatformType.PC },

new GameData { Name="Doodle Jump", YearOfRelease=2009, Rating=4,  
 Platform=PlatformType.Mobile },

new GameData { Name="Rocket League", YearOfRelease=2015, Rating=5,  
 Platform=PlatformType.PC },

};

}

}

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

<ListView ItemsSource="{Binding Games}">

</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:Paging.ViewModels"

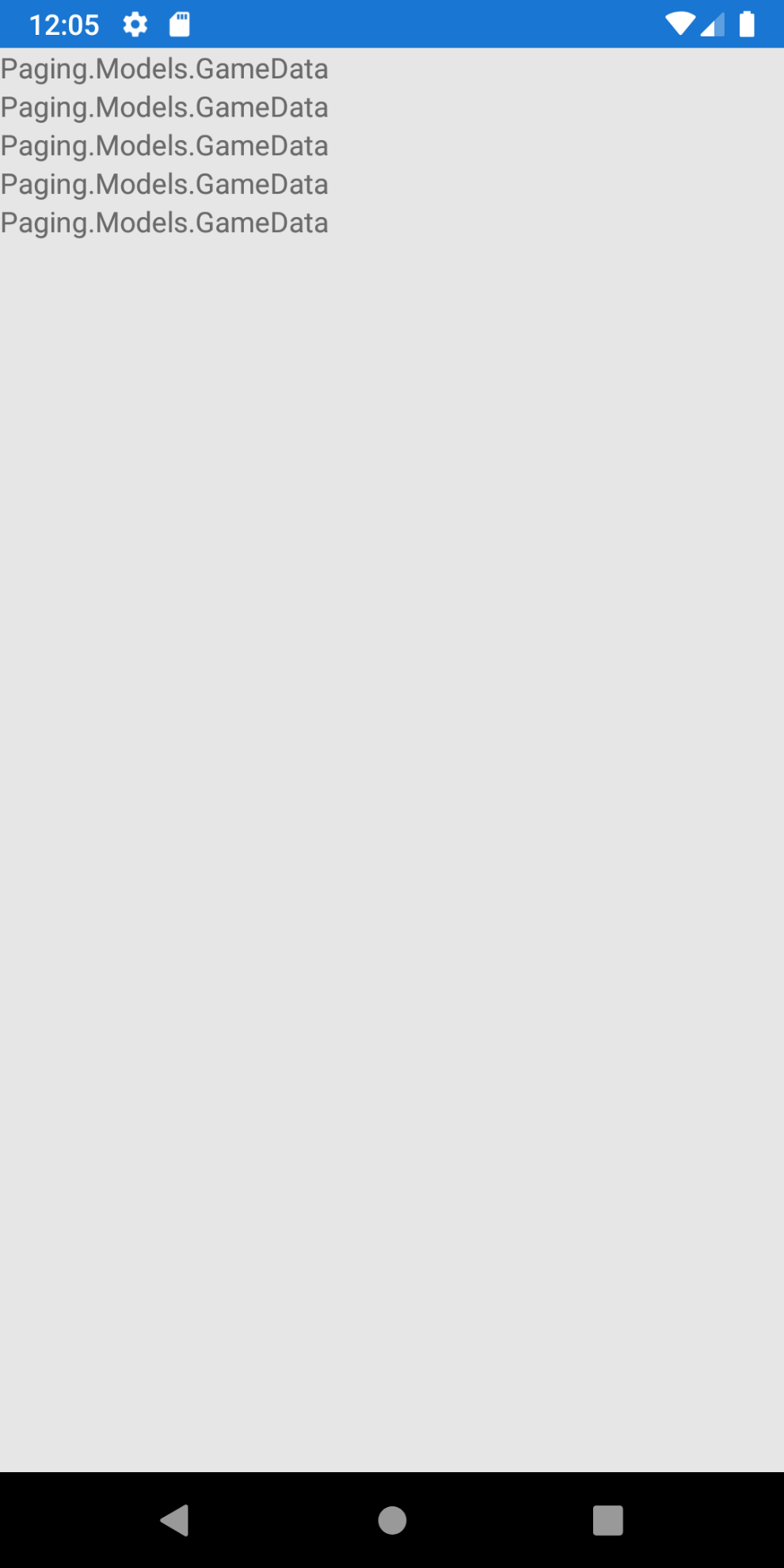
1. Just under that node and before the CollectionView add the code to set the BindingContext.

<ContentPage.BindingContext>

<vm:MainPageViewModel />

</ContentPage.BindingContext>

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



1. We are going to have our CollectionView just be the basic name of the game. To do this you can either add the ItemTemplate to the CollectionView like the previous ICE, or just add the ToString method to the GameData class

public override string ToString()

{

return Name;

}

1. To the CollectionView, set the **SelectionMode** to Single and then also add a property for the **SelectionChanged** event and just use the default values. This will make the method in the MainPage.xaml.cs file. Now when you select one of the games, this method will get called with the **GameData** object that was tied to that item.

From here we can now go and work on the second page.

## Part 2 – DetailPage Setup

1. At the project level, right click and Add and new item to the project. Make this a **ContentPage** and name it **DetailPage**. If you want to rename a page that got messed up, it is easier to just delete it and make a new one since the xaml and the code behind would have to be edited manually.
2. In the constructor for the DetailPage.xaml.cs, we will want to create the ViewModel here so that we can pass data into it. Add a parameter to the constructor that takes a **GameData** object. After the InitializeComponents call, create and assign the **BindingContext** to a new instance of **DetailPageViewModel** and pass the **GameData** into it. You might have to go back to the GameData class and make it public as well as the enum for the **PlatformType**.  
     
    BindingContext = new DetailPageViewModel(data);
3. Go back to the **DetailPageViewModel** class and add a constructor that takes the **GameData** as a parameter. Create a property for the **GameData** and save it. We will be using this to databind to controls on the screen to show the data.  
     
    public GameData Data { get; set; }

public DetailPageViewModel(GameData data)

{

Data = data;

}

1. The last thing that we need to do is to create multiple controls on the screen to display our data. These can all be just labels inside a StackLayout control with their text fields set to a {Binding Name} or other property.  
     
   Since the ViewModel is saving the data in a property that is named Data (or whatever you make it), we will have to use that property for the databinding. This means that you can bind to something like the name like this:  
     
    <Label Text="{Binding Data.Name}" />

Do this for all the data in the model. You can even data bind to an enum and it should print out the name of the enum.

## Part 3 – Navigation

At this point we have our data coming from the ViewModel, we are displaying a list of data and we can select 1 and get into the method that was created to handle item selection. Now we need to call the next page and give it the data that we need to show.

1. The first thing that we can do is add a call to the Navigation system to create and display the next page. In your CollectionView\_SelectionChanged method add a call using the following:  
     
    GameData selected = (GameData)e.CurrentSelection.First();

Navigation.PushAsync(new DetailPage(selected));

You need to cast the data from an object to the correct data type. The e.CurrectSelection is defined as readonly list of objects.  
  
The Navigation call will allow you to create the new page and to pass data to it. However, since we are not in a NavigationPage on the first one, this will fail.

1. Go back to the App.xaml.cs file and change the assignment of the MainPage to include a NavigationPage:

MainPage = new NavigationPage(new MainPage());

Now we have the first page letting us select an item on the list and have that data being passed into a second page. The navigation itself does not take long to add but prep work for it was a lot longer.

## Part 3 – Expanding GameData

From here, the app works well. Add a new field or two to the data model and get that information to display on the detail page. You can also add more information to the CollectionView instead of just the name of the game. This means data binding to the **GameData** object properties directly.

## Part 4 – Adding More Pages

Let us now add one more page for navigation from the main game list.

1. Create a new ContentPage and name it AboutPage. Add some information, such as your name to the about page. Clean up the rest of the data being added by default to have it be relevant for this app.
2. Go to the MainPage and add a Button above the CollectionView and add the Clicked property and create an event handler for the click (all from the xaml). Set the Text of the button to About.
3. In the new eventhandler, add the code to navigate to that about page.

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