Day 18: ListView and Adapters, Part 1

Over the past few days we have gotten to know a lot about Android and Xamarin.Android. Now lets us dive into the most commonly used View in Android of all ListView.

ListViews are found in pretty much every single application, in fact I challenge you real quick to name 3 apps on your phone which don’t have lists of data of some sought (excluding games). Not easy isn’t it?!. So let’s learn this integral part of Android Development in detail.

ListView as the name indicates is used to hold lists (or collections) of data in rows that can be scrolled through. These rows can be really simple and can be as complex as you want to be, each of these rows can contain any ViewGroups that we discussed during earlier days of this series of blog posts.

Each ListView gets its data through an **Adapter**. An Adapter is a class that is the glue between ListView and data that ListView shows. Adapter is also responsible to create and populate the rows of information that ListView shows. There are multiple types of Adapaters, like –

1. ArrayAdapter
2. SimpleAdapter
3. BaseAdapter
4. CursorAdapter

Let’s get into code and see how a simple ListView in Xamarin.Android looks like. Let’s first look at the AXML file –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent">  <ListView  android:minWidth="25px"  android:minHeight="25px"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:id="@+id/moviesListView" />  </LinearLayout> |

The AXML file is as simple as just adding the ListView to the parent ViewGroup, in our case, LinearLayout.

Gist file link: <https://gist.github.com/vkoppaka/7fc8d7a46ecfcf11bcdc>

Let’s take a look at the activity code –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace AdapterDemo1  {  [Activity(Label = "AdapterDemo1", MainLauncher = true, Icon = "@drawable/icon")]  public class MainActivity : Activity  {  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  // Set our view from the "main" layout resource  SetContentView(Resource.Layout.Main);  var moviesListView = FindViewById<ListView>(Resource.Id.moviesListView);  var moviesAdapter = new ArrayAdapter<Movie>(this, Android.Resource.Layout.SimpleListItem1, MoviesRepository.Movies);    moviesListView.Adapter = moviesAdapter;  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/26ecfc687e6d2d659a6e>

The two most important lines in the activity code are

1. Where we are creating an ArrayAdapter of Movie Data: An Array adapter is one of concrete implementations of BaseAdapater in Android and expects an array list of data. The display of ArrayAdapter is generally obtained by calling ToString() method on the object that Adapter deals with.
2. Finding ListView and assigning its adapter to the Movie Adapter

Let’s take a look the Movie Class and Movie Repository to round up all code classes

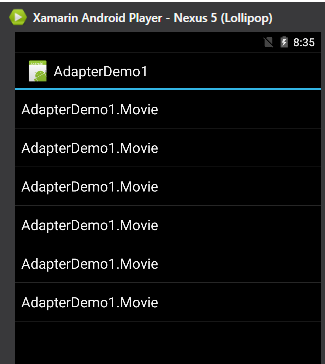
|  |
| --- |
| using System;  namespace AdapterDemo1  {  public class Movie  {  public string Title { get; set; }  public string Episode { get; set; }  public string Director { get; set; }  public DateTime ReleaseDate { get; set; }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  namespace AdapterDemo1  {  public static class MoviesRepository  {  public static List<Movie> Movies { get; private set; }  static MoviesRepository()  {  Movies = new List<Movie>();  AddMovies();  }  private static void AddMovies()  {  Movies.Add(new Movie  {  Title = "A New Hope",  Director = "George Lucas",  Episode = "IV",  ReleaseDate = new DateTime(1977, 05, 25)  });  Movies.Add(new Movie  {  Title = "The Empire Strikes Back",  Director = "George Lucas",  Episode = "V",  ReleaseDate = new DateTime(1980, 05, 17)  });  Movies.Add(new Movie  {  Title = "Return of the Jedi",  Director = "George Lucas",  Episode = "VI",  ReleaseDate = new DateTime(1983, 05, 25)  });  Movies.Add(new Movie  {  Title = "The Phantom Menace",  Director = "George Lucas",  Episode = "I",  ReleaseDate = new DateTime(1999, 05, 19)  });  Movies.Add(new Movie  {  Title = "Revenge of the Sith",  Director = "George Lucas",  Episode = "III",  ReleaseDate = new DateTime(2005, 05, 19)  });  Movies.Add(new Movie  {  Title = "The Force Awakens",  Director = "J.J. Abrams",  Episode = "VII",  ReleaseDate = new DateTime(2015, 12, 11)  });  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/45441b81f8bbe83be617>

This is pretty much straight .NET where we have a POCO object and a Demo-Need implementation of Repository Pattern.

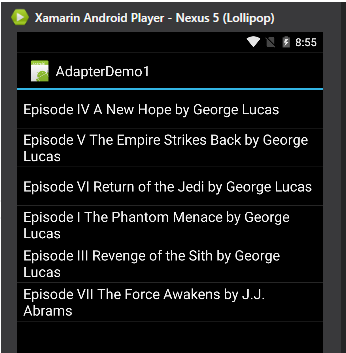
Let’s run this app and see how it looks –



Hmm, I don’t quite see any movie information rather I see the Movie Class’ namespace. Why is that? The reason is pretty simple, as I mentioned above, the display of ArrayAdapter is obtained by simply calling .ToString() on the object it deals with. So let’s try to override that behavior for our Movie object and show a better display.

|  |
| --- |
| using System;  namespace AdapterDemo1  {  public class Movie  {  public string Title { get; set; }  public string Episode { get; set; }  public string Director { get; set; }  public DateTime ReleaseDate { get; set; }  public override string ToString()  {  return "Episode " + Episode + " " + Title + " by " + Director;  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/c21eed2e27ea881ddae0>



That is more like it. And yes, for you all us eager Star Wars fans, I included Episode VII in the list as well ☺

# Detecting Item Click

Now let’s see how we go about detecting when an Item is clicked.

ListView exposes an event handler whenever an item is clicked, to register to the event just subscribe to the ItemClick Event Handler

|  |
| --- |
| moviesListView.ItemClick += moviesListView\_ItemClick; |

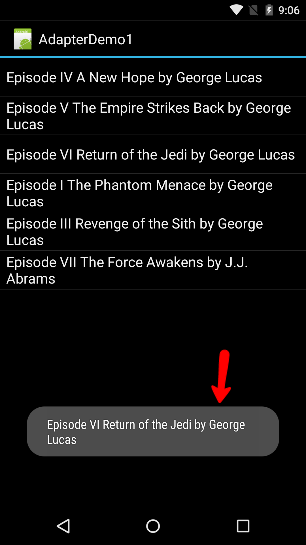
And the method implementation would be -

|  |
| --- |
| void moviesListView\_ItemClick(object sender, AdapterView.ItemClickEventArgs e)  {  Toast.MakeText(this, MoviesRepository.Movies[e.Position].ToString(), ToastLength.Long).Show();  } |

Gist file link: <https://gist.github.com/vkoppaka/0acc7e1b0ef4e600b977>

The key here is that the ItemClickEventArgs class a “Position” property that is used to identify what row is clicked. Don’t worry about Toasts code much, we will go into that detail in the upcoming blog posts.

So if we were to run this application and click and Item we should see something like –



There you go, a simple toast showing detail of the item clicked. Now, the possibilities on what we \*can\* do when an item is click is endless, I just happen to choose the most easy way to showcase what can be done.

That’s it for today, and tomorrow let’s dig into more on how to take more control on ListView Adapter and Layout.

Venkata