Day 21: RecyclerView

Today, we will explore more about RecyclerView in Xamarin.Android.

RecyclerView was introduced in Android 5.0 (Lollipop). RecyclerView is the new and better way to display large collections of data in Android. RecyclerView enforces the ViewHolder pattern Day 19 for ListView to reuse Views effectively, making it a very suitable View for handling large sets of data. On the other hand, RecyclerView, unlike ListView, does not provide any built in support for Item Click events, it is up to us to define these events using the .NET Events to achieve the same behavior of ListView.

RecyclerView similar to its other Android 5.0 counterparts exists in a Support Library for Xamarin.Android. To add RecyclerView to your Xamarin.Android application, please run the following command in your PowerShell (or you can use Nuget’s UI as well –

|  |
| --- |
| Install-Package Xamarin.Android.Support.v7.RecyclerView |

# Core Components

There are three major core components of a RecyclerView –

1. Adapter
2. LayoutManager
3. ViewHolder

## Adapter

An Adapter in RecyclerView, conceptually, is very similar to the Adapters we have looked so far. They are used to inflate item row layouts with the data that is bound to RecyclerView. But, Adapter in RecyclerView inherits from “**RecyclerView.Adapter**” as opposed to just BaseAdapter. The RecyclerView.Adapter exposes two overridable methods and a property that needs to be implemented.

* **OnBindViewHolder** – The OnBindViewHolder is the method that is responsible to bind the data that is passed to the RecyclerView at a given position. This method uses the ViewHolder that we will be creating in the OnCreateViewHolder method
* **OnCreateViewHolder** – The OnCreateViewHolder method is responsible for inflating the item layout row. This method also returns the ViewHolder from the inflated layout that is used by OnBindViewHolder
* **ItemCount** – Similar to ItemCount in regular Adapter, this property exposes the total number of rows in the given dataset.

The Adapter is also responsible for exposing an ItemClick event, but, let’s take a look at that after our initial RecyclerView sample is completed.

## LayoutManager

A LayoutManager in RecyclerView is responsible for positioning View items. RecyclerView comes with few out of the box Layout Managers to show the items in different formats –

* **LinearLayoutManager** – The LinearLayoutManager, as you would expect from the name, lays out the items in a linear fashion. LinearLayoutManager also supports Orientation property to position the views Horizontally or Vertically. There is also support for reversing a given list.
* **GridLayoutManager** – The GridLayoutManager lays out the items in a Grid format. You can define the number of spans that are the number of columns, which are supposed to be shown on the screen. GridLayoutManager supports Orientation property to position the views Horizontally or Vertically. There is also support for reversing a given list.
* **StaggeredGridLayoutManager** – The StaggeredGridLayoutManager lays out the items in a staggered format. You can defined the number of spans that are the number of columns, which are supposed to be shown on the screen. StaggeredGridLayoutManager supports Orientation property to position the views Horizontally or Vertically.

## ViewHolder

ViewHolder is used to look up and store references of the View items that form the item row. The ViewHolder works in conjunction with the Adapter to expose the ItemClick event for RecyclerView. A ViewHolder inherits from the “**RecyclerView.ViewHolder**” class and finds the Views using the familiar FindViewById<T> method.

# Code

Let’s take a look at all of these core components put together in code and see a RecyclerView in action.

First, let’s start with the Main.axml file –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:local="http://schemas.android.com/apk/res-auto"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent">  <include  android:id="@+id/toolbar"  layout="@layout/toolbar" />  <LinearLayout  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:id="@+id/main\_content"  android:layout\_below="@id/toolbar">  <android.support.v7.widget.RecyclerView  android:id="@+id/recyclerView"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content" />  </LinearLayout>  </RelativeLayout> |

Gist file link: <https://gist.github.com/vkoppaka/2a7a61e8069cfcb9ad7a>

If you notice carefully, we added RecyclerView to the Layout. The RecyclerView as I mentioned above is in the v7 support library namespace.

Next up, let’s look at the MovieRow.axml file, this is very similar to the CardView example we saw in yesterday’s blogpost –

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:cardview="http://schemas.android.com/apk/res-auto"  android:orientation="vertical"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:gravity="center\_horizontal"  android:padding="5dp">  <android.support.v7.widget.CardView  android:layout\_width="fill\_parent"  android:layout\_height="245dp"  android:layout\_gravity="center\_horizontal"  cardview:cardElevation="4dp"  cardview:cardCornerRadius="5dp">  <LinearLayout  android:layout\_width="fill\_parent"  android:layout\_height="240dp"  android:orientation="vertical"  android:padding="8dp">  <ImageView  android:layout\_width="fill\_parent"  android:layout\_height="190dp"  android:id="@+id/imageView"  android:scaleType="centerCrop" />  <TextView  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:textAppearance="?android:attr/textAppearanceMedium"  android:textColor="#333333"  android:text="Photo Title"  android:id="@+id/cardViewText"  android:layout\_gravity="center\_horizontal"  android:layout\_marginLeft="5dp" />  </LinearLayout>  </android.support.v7.widget.CardView>  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/5c8bfd54f9c8760e09af>

Next up, let’s take a look at the ViewHolder that will be used to find and hold references of the MovieRow Layout –

|  |
| --- |
| using System;  using Android.Support.V7.Widget;  using Android.Views;  using Android.Widget;  namespace RecyclerViewSample  {  public class MovieViewHolder : RecyclerView.ViewHolder  {  public TextView MovieNameTextView { get; set; }  public ImageView MovieImageView { get; set; }    public MovieViewHolder(View itemView, Action<int> listener) : base(itemView)  {  MovieNameTextView = itemView.FindViewById<TextView>(Resource.Id.cardViewText);  MovieImageView = itemView.FindViewById<ImageView>(Resource.Id.imageView);  itemView.Click += (s,e) => listener(Position);  }  }  } |

The ViewHolder’s constructor is used to find the items using the FindViewById call and this class is also responsible for handling ItemClick methods. The second parameter is an Action that can listen and pass a position to whoever that calls it.

Now, let’s take a look at the Adapter –

|  |
| --- |
| using System;  using System.Collections.Generic;  using Android.Support.V7.Widget;  using Android.Views;  namespace RecyclerViewSample  {  public class MovieAdapter : RecyclerView.Adapter  {  public event EventHandler<int> ItemClick;  private readonly List<Movie> movies;  public MovieAdapter(List<Movie> movies)  {  this.movies = movies;  }  public override void OnBindViewHolder(RecyclerView.ViewHolder holder, int position)  {  var movieViewHolder = (MovieViewHolder)holder;  movieViewHolder.MovieNameTextView.Text = movies[position].Title;  movieViewHolder.MovieImageView.SetImageResource(Resource.Drawable.starwarslogo);  }  public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int viewType)  {  var layout = LayoutInflater.From(parent.Context).Inflate(Resource.Layout.MovieRow, parent, false);  return new MovieViewHolder(layout, OnItemClick);  }  public override int ItemCount  {  get { return movies.Count; }  }  void OnItemClick(int position)  {  if (ItemClick != null)  ItemClick(this, position);  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/c777aadd0af9171a5126>

The OnCreateViewHolder is used to inflate the layout and OnBindViewHolder is used to Bind the data. There is an EventHandler for ItemClick that is defined by the Adapter as well, which passes a listener to the ViewHolder.

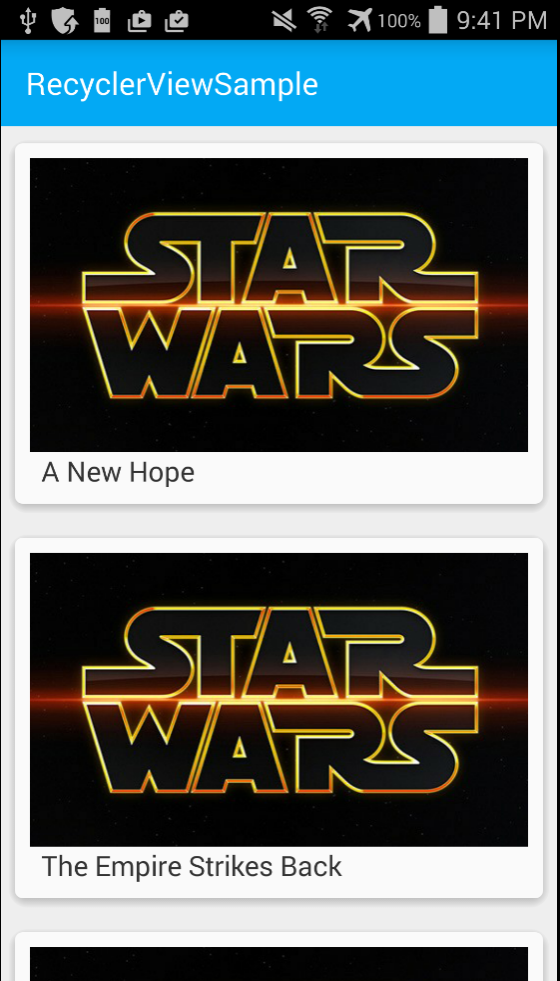
Finally, to piece all these things together, let’s take a look at the Activity –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Support.Design.Widget;  using Android.Support.V7.Widget;  using Android.Widget;  namespace RecyclerViewSample  {  [Activity(Label = "RecyclerViewSample", MainLauncher = true, Icon = "@drawable/icon")]  public class MainActivity : BaseActivity  {  protected override int LayoutResource  {  get { return Resource.Layout.main; }  }  private RecyclerView recyclerView;  private RecyclerView.LayoutManager layoutManager;  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  recyclerView = FindViewById<RecyclerView>(Resource.Id.recyclerView);    layoutManager = new LinearLayoutManager(this, LinearLayoutManager.Vertical, false);  recyclerView.SetLayoutManager(layoutManager);  var moviesAdapter = new MovieAdapter(MoviesRepository.Movies);  recyclerView.SetAdapter(moviesAdapter);  moviesAdapter.ItemClick += MoviesAdapter\_ItemClick;  SupportActionBar.SetDisplayHomeAsUpEnabled(false);  SupportActionBar.SetHomeButtonEnabled(false);  }  private void MoviesAdapter\_ItemClick(object sender, int e)  {  var linearLayout = this.FindViewById<LinearLayout>(Resource.Id.main\_content);  Snackbar.Make(linearLayout, MoviesRepository.Movies[e].ToString(), Snackbar.LengthLong).Show(); // Don’t forget to show!  }  }  } |

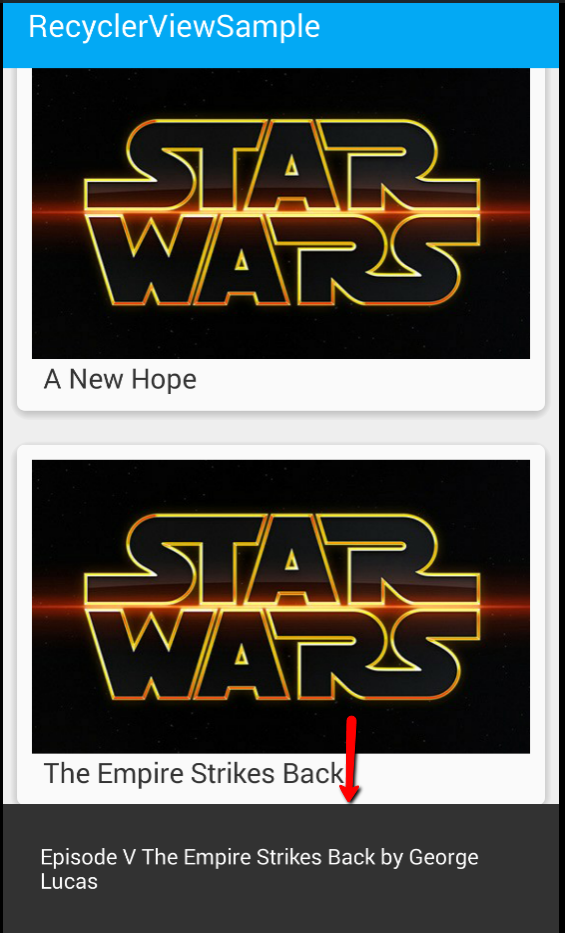
Gist file link: <https://gist.github.com/vkoppaka/61cea001216330f8c8bb>

In the Activity file, we are defining a LayoutManager, in this scenario a LinerLayoutManager with vertical orientation. Then, we are calling the RecyclerView’s SetLayoutManager method to set the LayoutManager and finally, we create an instance of MovieAdapter and call the RecyclerView’s SetAdapter method to assign the adapter to the view.

If you were to run the application, you would see not many differences from a UI point of view from yesterday, but under the covers, we are now using RecyclerView.



And if you click an item in the list –



We now show the Toast(SnackBar, more on SnackBars later) of the selected item.

That’s it for today, see you all tomorrow.