Day 16: Controls: Media Controls

Today we will take a look at different Media Controls in Xamarin.Android. We will explore the following controls –

1. Gallery
2. ImageView
3. ImageButton

# Gallery

Gallery control in Android is used to showing items in a horizontally scrolling list. Important thing to note here is in recent versions of Android, this control has become Obsolete in favor of HorizontalScrollView and ViewPager. But, if you are targeting older Android Devices, Gallery is still a really good option to show horizontally scrolling lists.

To get started, drag and drop a Gallery Control on to the Main.axml (or any other activity) 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"  android:minWidth="25px"  android:minHeight="25px">  <Gallery  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/gallery" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/35bf7dccc1f886b872c2>

And as always, to find the Gallery in the Activity file, we use the FindViewById method –

|  |
| --- |
| Gallery gallery = FindViewById<Gallery>(Resource.Id.gallery); |

Now, in order to set the data source (adapter) for Gallery widget, we need to first define an adapter that gets list of images to show. For this example, I am using pictures of cat (who doesn’t like cute cat pictures?) that are being served from the Resources folder, but there is no stopping you, Dear reader, to make this list be driven from something over a REST API call.

So, let’s take a look at how the Adapter looks like that gets list of picture to show –

|  |
| --- |
| using Android.Content;  using Android.Views;  using Android.Widget;  namespace FunWithMediaControls  {  public class CuteCatsAdapter : BaseAdapter  {  Context context;  public CuteCatsAdapter (Context c)  {  context = c;  }  public override int Count { get { return imagesToShow.Length; } }  public override Java.Lang.Object GetItem (int position)  {  return null;  }  public override long GetItemId (int position)  {  return 0;  }  // create a new ImageView for each item referenced by the Adapter  public override View GetView (int position, View convertView, ViewGroup parent)  {  ImageView i = new ImageView (context);  i.SetImageResource (imagesToShow[position]);  i.LayoutParameters = new Gallery.LayoutParams (500, 500);  i.SetScaleType (ImageView.ScaleType.FitXy);  return i;  }  // references to our images  int[] imagesToShow = {  Resource.Drawable.cat1,  Resource.Drawable.cat2,  Resource.Drawable.cat3  };  }  } |

Gist file: <https://gist.github.com/vkoppaka/84c163d8b8086d50807f>

This is just like any other Adapter we have seen so far, just that we are returning images using the “imagesToShow” array from Resources.Drawable ids. The GetView method, which generally inflates the layout, is used to put an ImageView (more on ImageView in this blogpost below) in the Gallery and the ImageView’s resource is set to the current image to be shown from the Resources Pictures array.

And, finally, we need to set the Adapter of the Gallery to our CuteCatsAdapter like –

|  |
| --- |
| gallery.Adapter = new CuteCatsAdapter(this); |

Let’s take a look at the full Activity code in action –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Widget;  namespace FunWithMediaControls  {  [Activity(Label = "FunWithMediaControls", 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);  Gallery gallery = FindViewById<Gallery>(Resource.Id.gallery);  gallery.Adapter = new CuteCatsAdapter(this);  }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/8b63bbfa7b8403718827>

If you were to run the application, you should see a horizontally scrollable list of cat images –



## Events

The most important event that Gallery Widget exposes is the ItemClick event. Let’s see Gallery Item Click in action –

|  |
| --- |
| gallery.ItemClick += Gallery\_ItemClick;    private void Gallery\_ItemClick(object sender, AdapterView.ItemClickEventArgs e)  {  Toast.MakeText(this, e.Position.ToString(), ToastLength.Short).Show();  } |

We can get the position of the currently selected item from e.Position.

# ImageView

The next Media View that we will be exploring is the ImageView. ImageView is used to show Images in your Android application. Let’s see a simple ImageView AXML –

|  |
| --- |
| <?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"  android:minWidth="25px"  android:minHeight="25px">  <ImageView  android:src="@drawable/cat1"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/imageView1" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/560ff580536548d3b07b>

The most important property that ImageView exposes is the **android:src** attribute which is used to set the image that supposed to be shown –

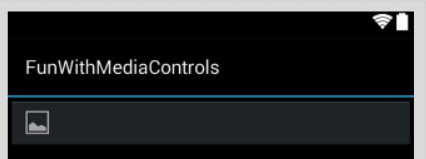


# ImageButton

Next up in Media Views in ImageButton. ImageButton as the name indicates, is used to show Images in a Button. Let’s take a look at ImageButton AXML –

|  |
| --- |
| <?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"  android:minWidth="25px"  android:minHeight="25px">  <ImageButton  android:src="@android:drawable/ic\_menu\_gallery"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/imageButton1"  android:scaleType="fitStart" />  </LinearLayout> |

Similar to the ImgeView the most important property that ImageButton exposes is the **android:src** attribute which is used to show the image that you picked –



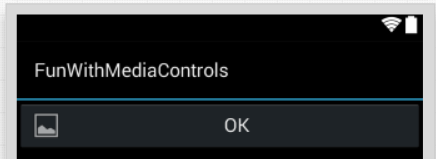
# Text and Image Button

The ImageButton view is used only to show buttons with Images in it, what if, if you want to show Image in a Button along with some text? For this purpose, we can just use the Button class that we have seen so many times till now, but we will use a new **android:drawableLeft**  attribute. Let’s see that in action –

|  |
| --- |
| <?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"  android:minWidth="25px"  android:minHeight="25px">  <Button  android:id="@+id/buttonok"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:drawableLeft="@android:drawable/ic\_menu\_gallery"  android:text="OK" />  </LinearLayout> |

Gist file link: <https://gist.github.com/vkoppaka/2b64f1d5e758da48c56c>

And the button would look like –



That’s it for the Xamarin.Android Controls mini-series. We will begin exploring more aspects of Android development in the upcoming posts. Stay tuned.