By Diptimaya Patra

# Xamarin Android Application Development

# Xamarin Android Application Development

Diptimaya Patra

This book is for sale at http://leanpub.com/xamarin-android-app-dev

This version was published on 2014-03-16

# Tweet This Book!

Please help Diptimaya Patra by spreading the word about this book on Twitter!

The suggested hashtag for this book is #XamarinAndroidBookDPatra.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#XamarinAndroidBookDPatra

# Contents

# Layout

Layout in android is a View group type, which can contain views inside it. So basically it is a container, based on requirement. Anything that can be a view can be a child of Layout. A custom layout could also be created for custom requirement, as the Layouts are derived from ViewGroup class.

There are few types of Layouts available in Android, we would discuss all.

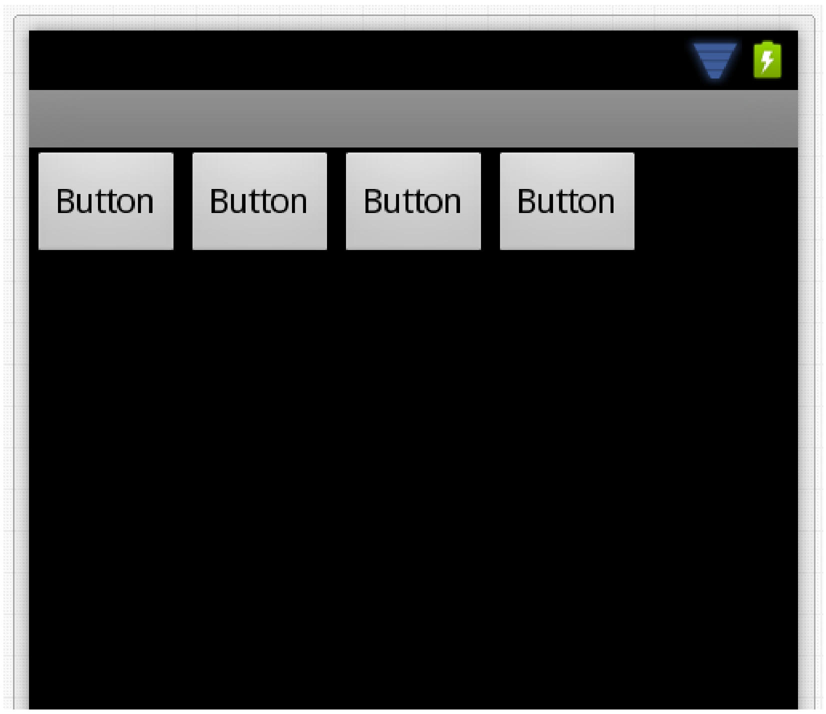
1. Linear Layout
2. Relative Layout
3. Table Layout
4. Frame Layout

## Linear Layout

A Linear Layout could have children displaying either in horizontal or in vertical. Let's have a layout example we would do for all the above layouts. We would have two EditText controls with describing TextView and a Button control to do something.

**Linear Layout – Horizontal**

As discussed above, a Linear Layout with orientation set as horizontal; places children views in horizontal order. Such as following.



The XML layout would be as below.

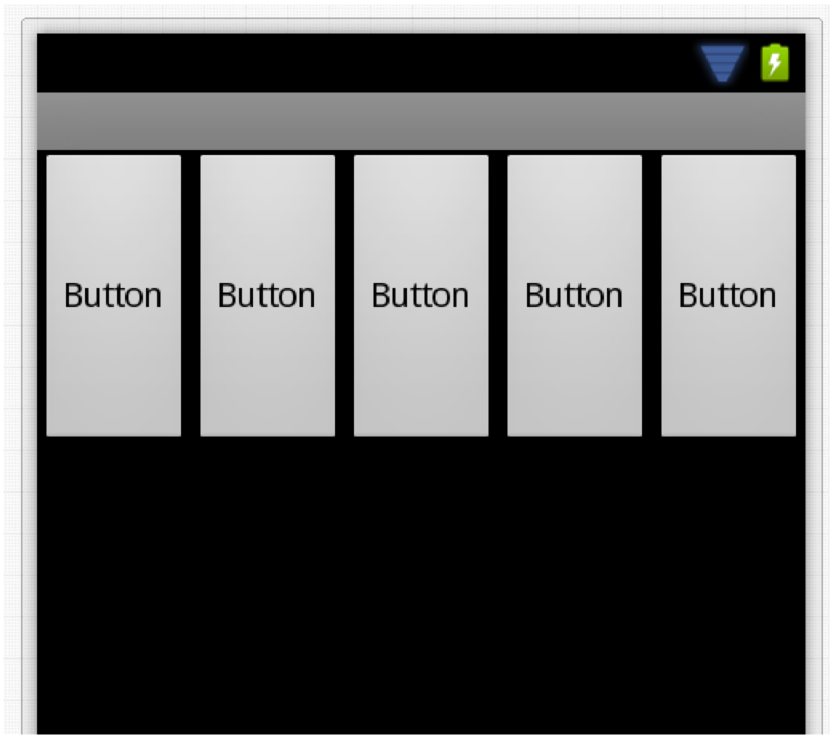
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:minWidth="25px"
    android:minHeight="25px">
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent" />
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent" />
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent" />
    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent" />
</LinearLayout>
```

As you see in the above layout XML, Linear Layout has a property as Layout_Height as wrap_-content. Basically we have four types by which we could set the width or height of the ViewGroup or the View.

1. Fill Parent – Fills the parent

2. Wrap Content – Wraps with the maximum allowed (Parent's) width or height
3. Match Parent – Same as Fill Parent but added with different OS version
4. Unit Values
5. DP – Density independent Pixels, based on physical density of the screen.
6. SP – Scale independent Pixels, scaled by user's font size. Recommended to use with fonts.
7. DPI – Same as DP
8. PX – Pixels, corresponds to actual pixels on the screen.
9. IN – Inches, based on the physical size of the screen.
10. MM – Millimetres, based on the physical size of the screen.
11. PT – Points, 1/72 of the inch based on the physical size of the screen.

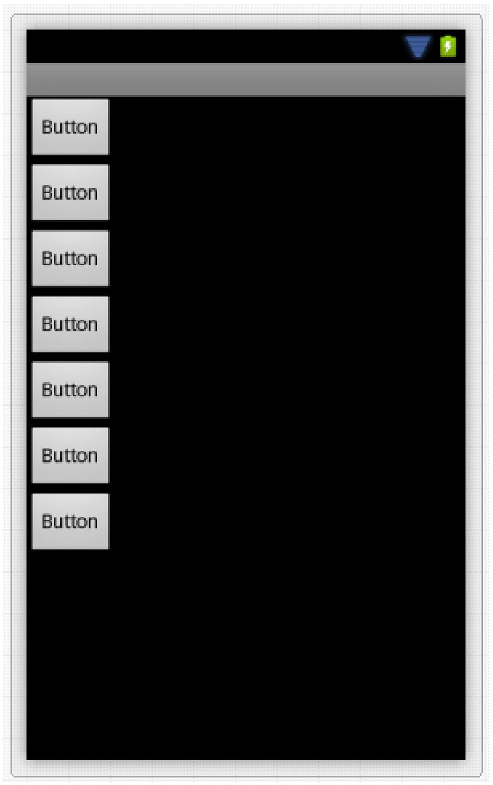
If we add few more buttons to the above layout, it would not scroll and the height would wrap.



For the above layout, we have 8 buttons; but displayed 5 buttons, also we cannot scroll. We would see later, how to add scrolling to the layout.

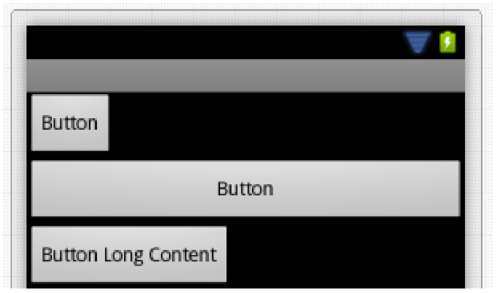
**Linear Layout – Vertical**

A Linear Layout with orientation set as vertical; places children views in vertical order. Such as following:



As you see in above image, the buttons are having Layout_Width property as Wrap_Content; that is the reason buttons are wrapped wherever content is finished. Let's add few more buttons and with different Layout_Width setting.

```
<Button
    android:text="Button"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent" />
<Button
    android:text="Button"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
<Button
    android:text="Button Long Content"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent" />
```



If you look at above Layout and it's button's Layout_Width property changes with respect to the content; you would understand it properly. These properties are common property for View or ViewGroup.

Now a Vertical Linear Layout cannot scroll it's content automatically. As mentioned earlier, we would discuss it in later chapter.

## Relative Layout

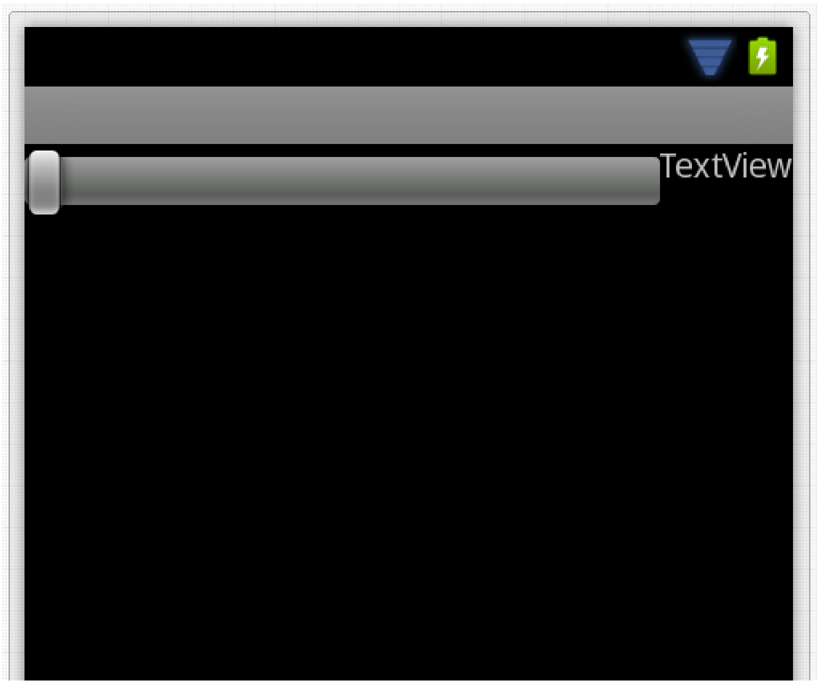
A Relative Layout has always relationship with its children. You can also say that, children of Relative Layout are related to layout. This is one of the complicated layouts you would ever see. The reason is too many properties to set for child and parent to get what we want. The following are the properties.

1. Relative to Parent Container
2. Layout_AlignParentBottom – displays child at bottom of parent
3. Layout_AlignParentLeft – displays child on the left side of the parent
4. Layout_AlignParentRight – displays child on the right side of the parent
5. Layout_AlignParentTop – displays child at the top of the parent
6. Layout_CenterHorizontal – centers the child horizontally in the parent
7. Layout_CenterInParent – centers the child both horizontally and vertically in the parent
8. Layout_CenterVertical – centers the child vertically in the parent
9. Relative to Other View
10. Layout_Above – places the child above the view
11. Layout_Below – places the child below the view
12. Layout_ToLeftOf – places the element left of the view
13. Layout_ToRightOf – places the element right of the view
14. Align with Other View
15. Layout_AlignBaseline – aligns baseline of the new view with the baseline of specified view
16. Layout_AlignBottom – aligns bottom of the new view with the bottom of the specified view
17. Layout_AlignLeft – aligns left of the new view with the left of the specified view
18. Layout_AlignRight – aligns right of the new view with the right of the specified view
19. Layout_AlignTop – aligns top of the new view with the top of the specified view

The following is a simple example where, a SeekBar control's current Progress property would be displayed on the right side.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="TextView" />
    <SeekBar
        android:id="@+id/seekBar1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@id/textView1" />
</RelativeLayout>
```

This would look similar to the following in design view.



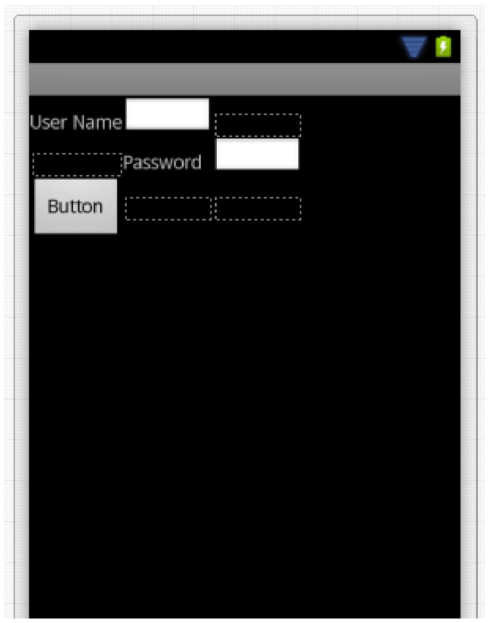
As it is a complex layout to describe, we would use it wherever it's useful and we would discuss the layout there.

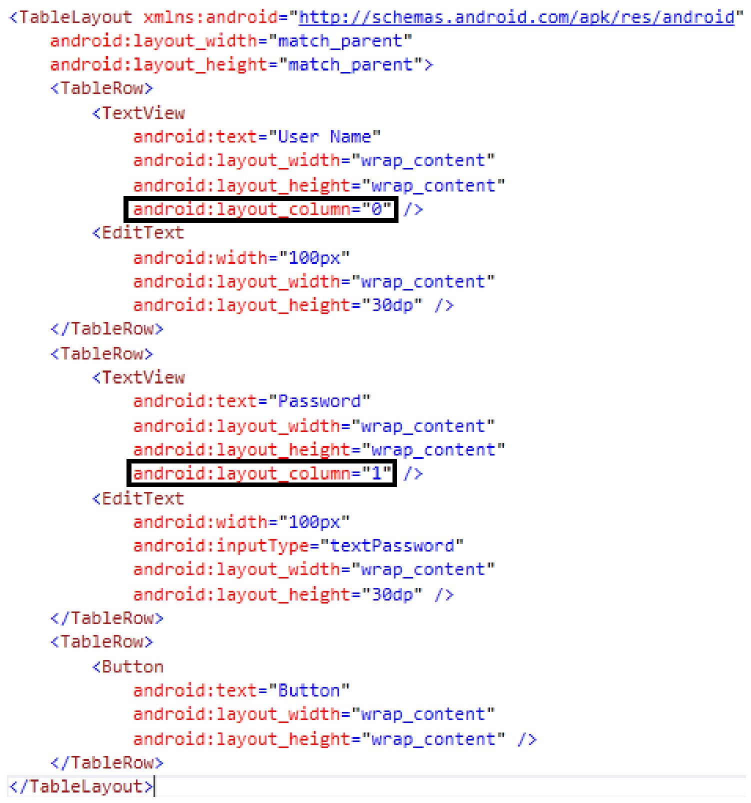
## Table Layout

This is one of the most used Layouts. As the name says the View is organised in Rows and Columns. However to have columns in the Row, we need to use a Table Row, which would contain the children. The number of children present in a TableRow, is exactly the number of Columns present in the Row.

Also, it’s not obvious that; the number of columns present in first row would be same in the following Row. That’s the poor thing about it; each row’s column count is dependent on its children count.

The following design is made using a Table Layout.

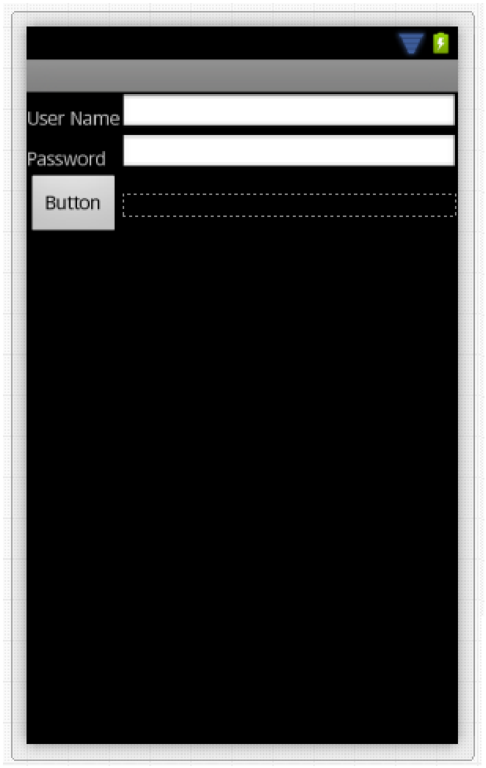


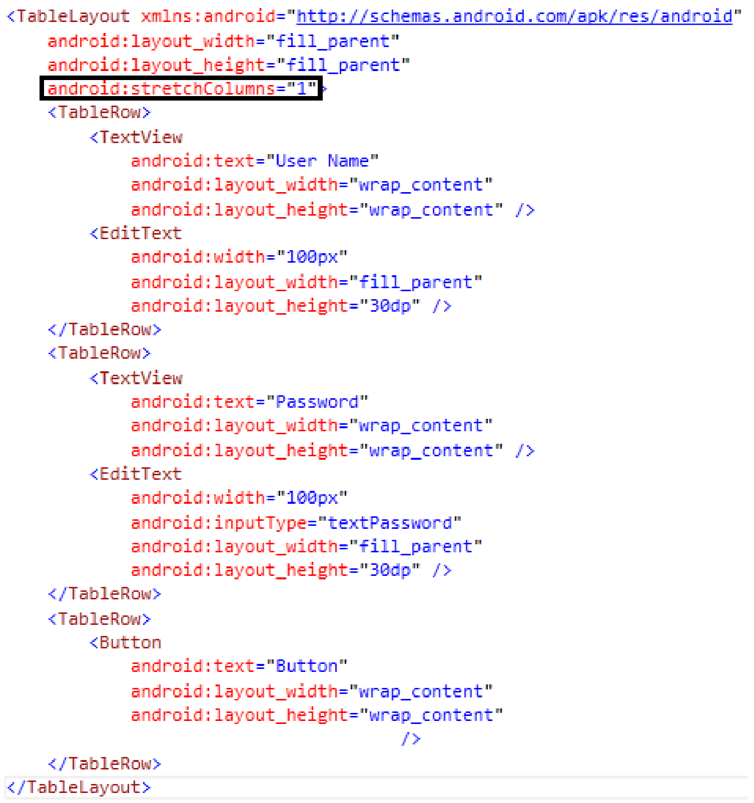


```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow>
        <TextView
            android:text="User Name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="0" />
        <EditText
            android:width="100px"
            android:layout_width="wrap_content"
            android:layout_height="30dp" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="Password"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="1" />
        <EditText
            android:width="100px"
            android:inputType="textPassword"
            android:layout_width="wrap_content"
            android:layout_height="30dp" />
    </TableRow>
    <TableRow>
        <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>
</TableLayout>
```

The above example shows that we can place elements in virtual columns. This is calculated based on the space available with respect to the maximum column assigned +1. So if we are assigning Layout_Column as 1 then it would be a 3 column row.

Now adjust the above view to look decent. The following changes are made.



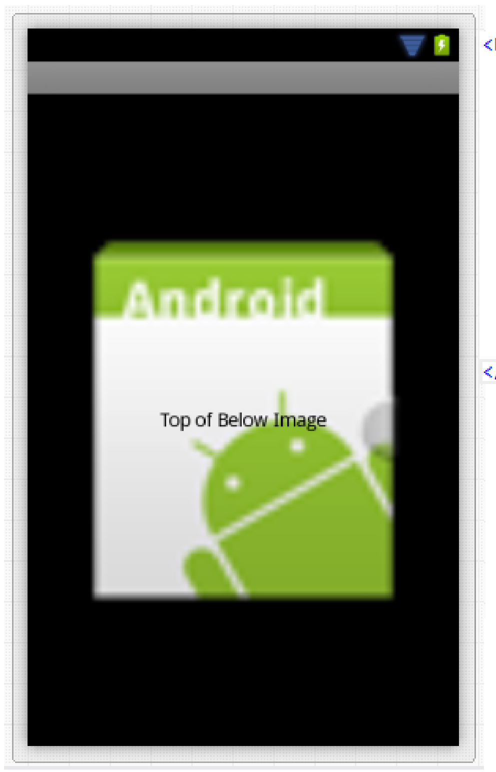


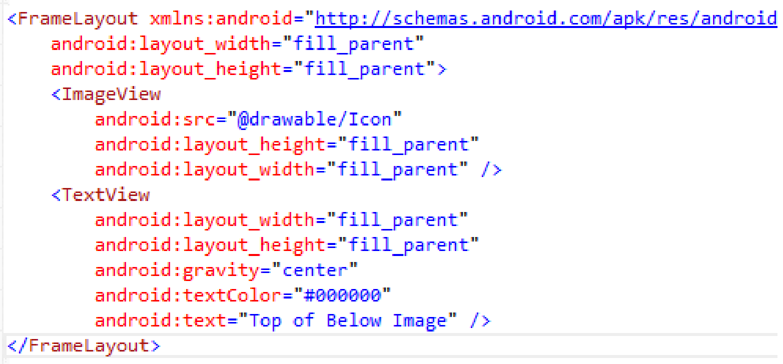
```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="User Name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <EditText
            android:width="100px"
            android:layout_width="fill_parent"
            android:layout_height="30dp" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="Password"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <EditText
            android:width="100px"
            android:inputType="textPassword"
            android:layout_width="fill_parent"
            android:layout_height="30dp" />
    </TableRow>
    <TableRow>
        <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
                                        />
    </TableRow>
</TableLayout>
```

The StretchColumns property of the Table itself fixed the first two rows. Likewise we would be discussing more about these properties while we progress in this book.

## Frame Layout

Each child view in Frame Layout starts from the Top Left of the Layout.



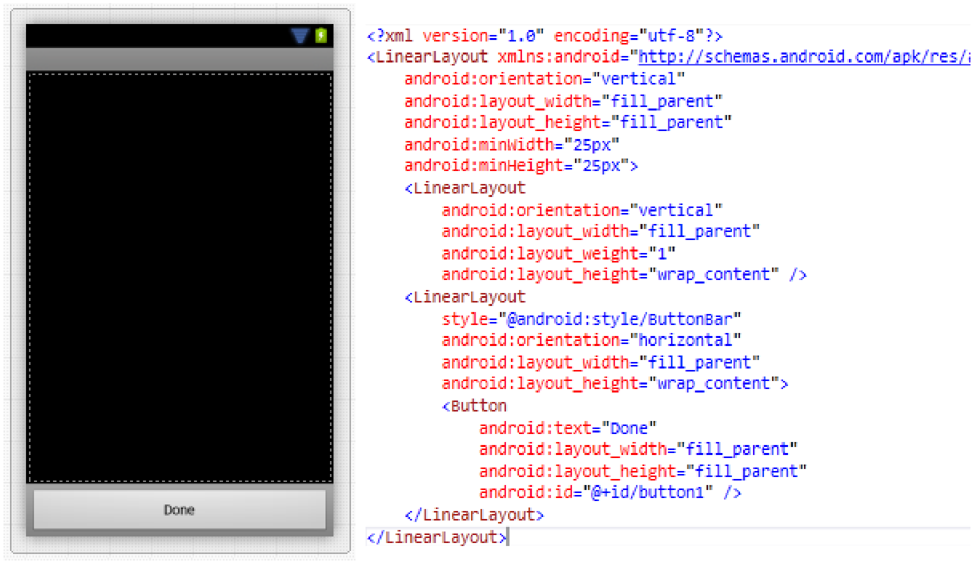

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageView
        android:src="@drawable/Icon"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:textColor="#000000"
        android:text="Top of Below Image" />
</FrameLayout>
```

As you see in the above design, the ImageView and TextView are layered one after another. So basically, each starts displaying from the top left of the parent layout. It's up to the designer to design it.

## Using Android Styles in Layout

Android has also some inbuilt styles for Layouts. In some design scenarios, where we would require a fixed layout at the bottom of the screen. The fixed layout may contain a set of controls. In this post, we would see how we do that.

Here is a sample layout with LinearLayout (above the last LinearLayout).



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/a
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:minWidth="25px"
    android:minHeight="25px">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_weight="1"
        android:layout_height="wrap_content" />
    <LinearLayout
        style="@android:style/ButtonBar"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <Button
            android:text="Done"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/button1" />
    </LinearLayout>
</LinearLayout>
```

As you see in above image, the last control before the Fixed Layout, should have the above properties with values.

Now, the Linear Layout would become fixed at the bottom, once you apply the style of ButtonBar.

Now the Button inside it is the content of the fixed layout.

## Conclusion

We have discussed the Layouts in Xamarin Android development. If you are a designer, you should be able to mix and match the Layouts to provide the perfect required design. Keep designing.

# ListView

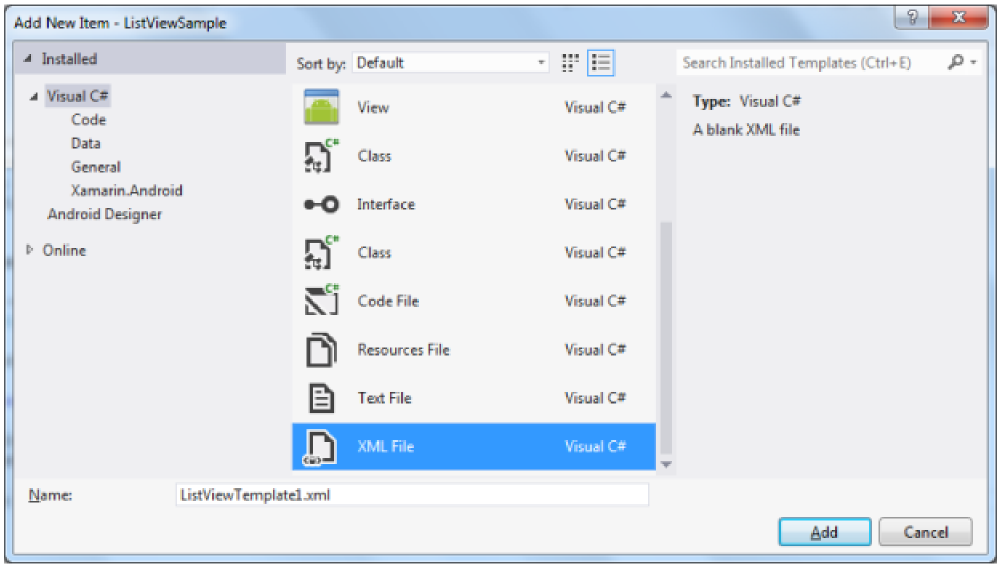
ListView is a ViewGroup that displays a list of items that could be scrollable. We had an interaction with a ListView using the ListActivity. We would see few more things about ListView in this chapter.

## Binding Data to ListView

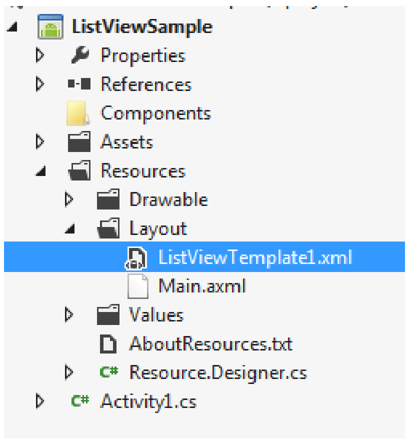
Previously we had an encounter with ListActivity which had a ListView, and we used a ListActivity for displaying data or collection. Here we would use theListView control and bind data or collection to it.

We would need a template for the ListView, so let's add one.

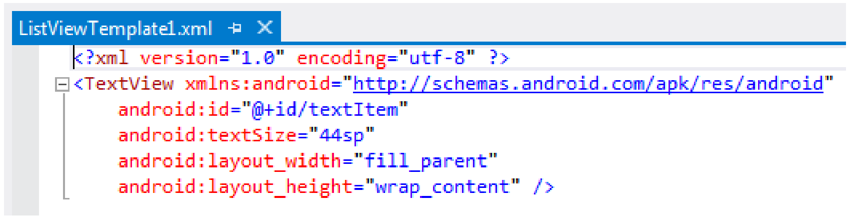
Add an XML file to the Layout folder.



Name: ListViewTemplate1.xml

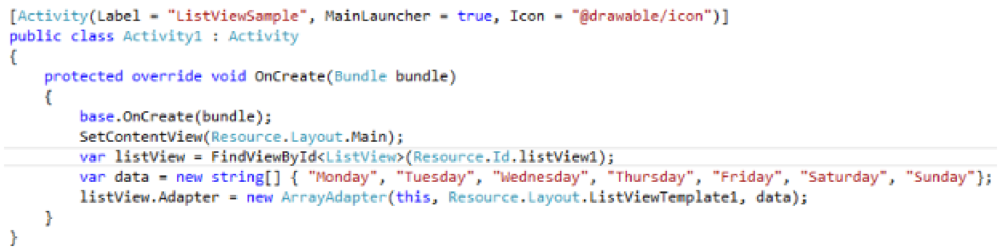


Add a TextView in the XML file. This is for ListView template.



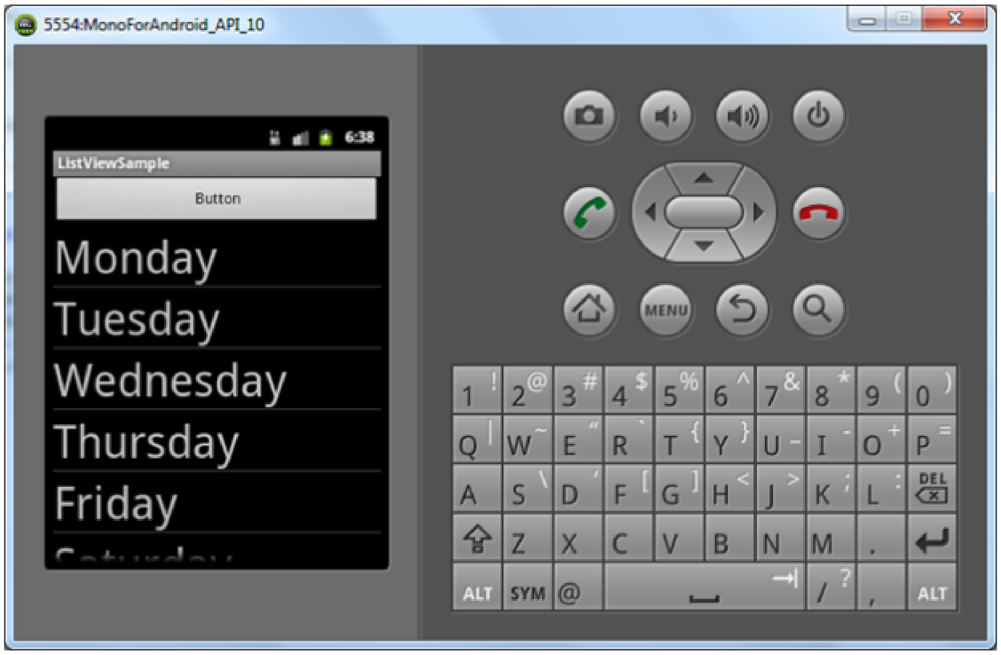
```xml
<?xml version="1.0" encoding="utf-8" ?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textItem"
    android:textSize="44sp"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

And as you see in the below image, the Activity1 is inherited from Activity class unlike our last post where we inherited from ListActivity. Rest of the code is self-explanatory. We are setting the Adapter property of the ListView with ArrayAdapter.



```csharp
[Activity(Label = "ListViewSample", MainLauncher = true, Icon = "@drawable/icon")]
public class Activity1 : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.Main);
        var listView = FindViewById<ListView>(Resource.Id.listView1);
        var data = new string[] { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};
        listView.Adapter = new ArrayAdapter(this, Resource.Layout.ListViewTemplate1, data);
    }
}
```

Now, let's run the application and see the ListActivity in action.



As you see, unlike ListActivity, we have other controls in the Activity.

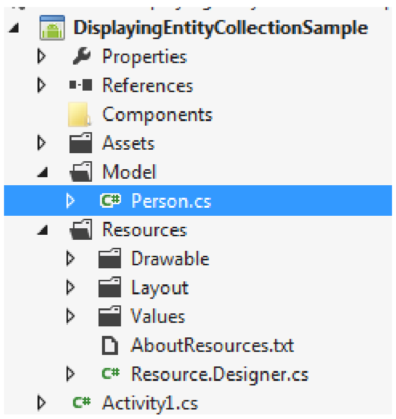
## Display Entity Collection

In real time, we would be displaying a collection of entity. Here, we would display a collection of some entity in ListView.

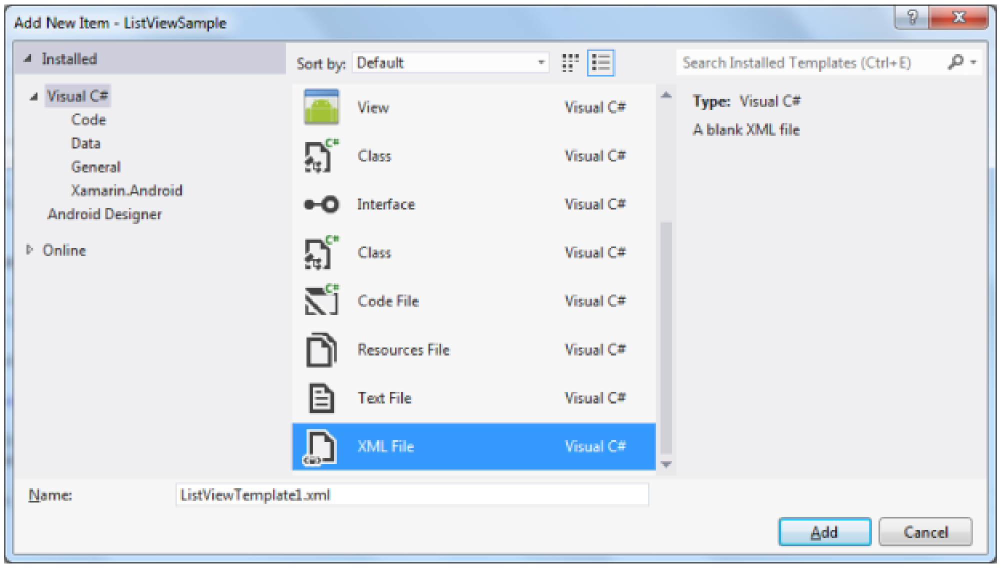
Let's start with creating the following Person entity.

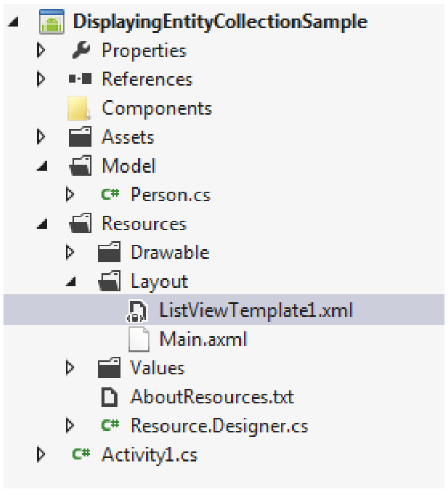
```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string EmailId { get; set; }
}
```

The Person entity class is located under Model.

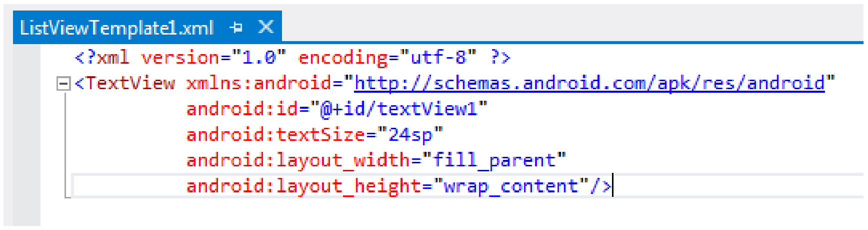


Now let's add the ListView template XML file under Layout folder.



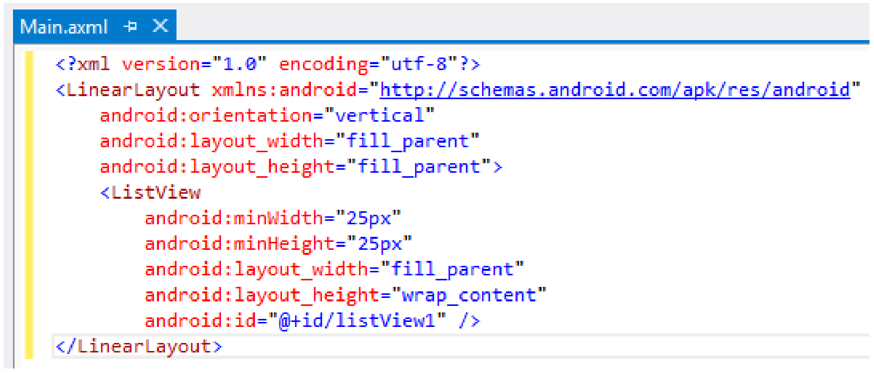


Keep a TextView for now inside the XML file.



```
<?xml version="1.0" encoding="utf-8" ?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/textView1"
          android:textSize="24sp"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"/>
```

Add a ListView control to the Main Layout.



```
<?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="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/listView1" />
</LinearLayout>
```

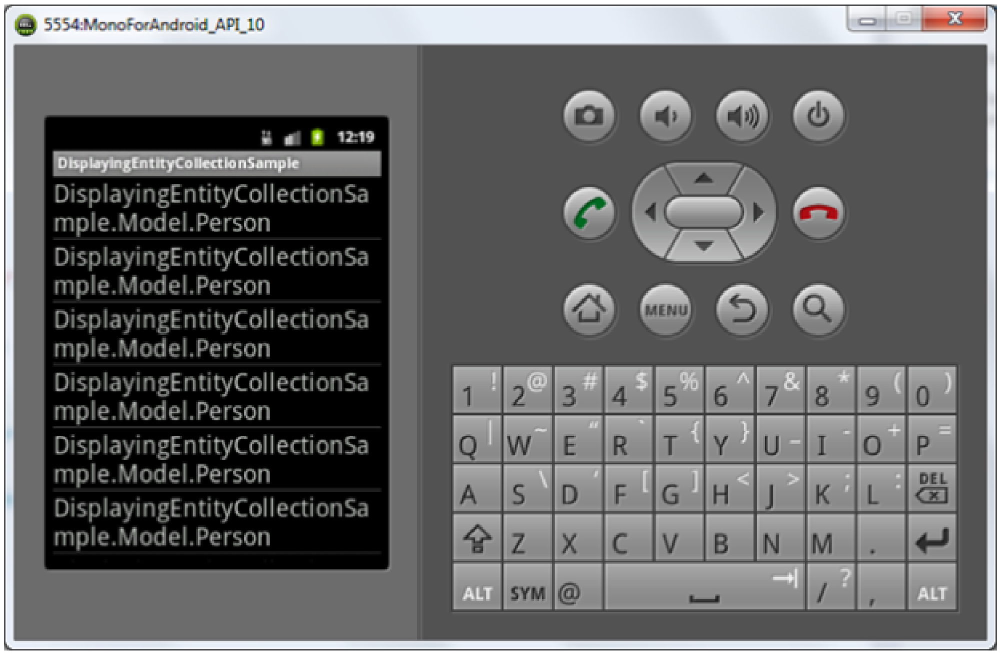
The following method would generate sample data.

```
private List<Person> CreateSampleData(int range)
{
    var persons = new List<Person>();

    for (int i = 1; i <= range; i++)
    {
        var person = new Person
        {
            FirstName = string.Format("FirstName{0}", i),
            LastName = string.Format("LastName{0}", i),
            EmailId = string.Format("FirstName{0}.LastName{0}@some.com", i)
        };
        persons.Add(person);
    }

    return persons;
}
```

Now in Activity1, we would get the ListView control and set the data by using following code.

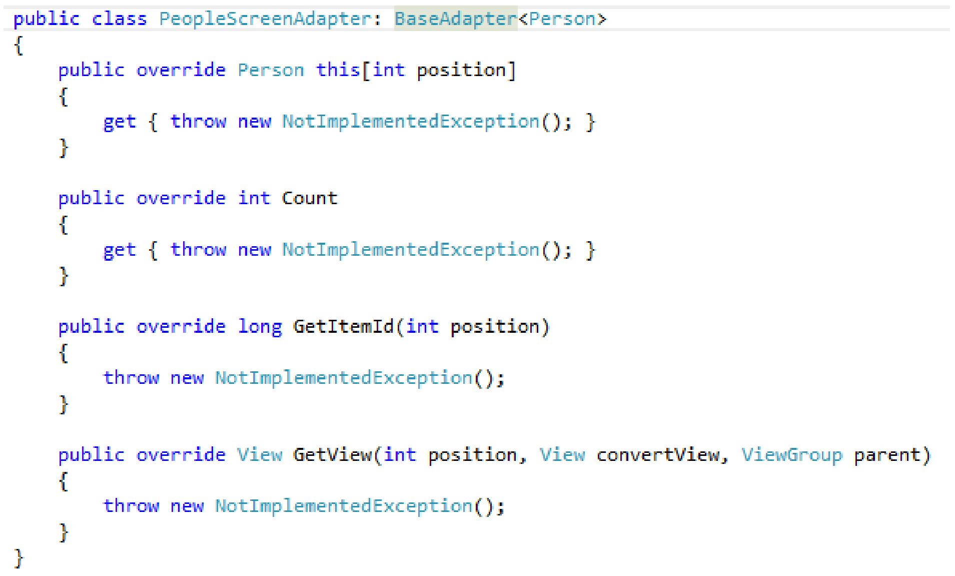
```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);
    var listView = FindViewById<ListView>(Resource.Id.listView1);

    var data = CreateSampleData(20);
    listView.Adapter = new ArrayAdapter(this, Resource.Layout.ListViewTemplate1, data);
}
```

Let’s run the application. Oops, the entity class name is displayed instead of data. That means we need to do a different way to display data.



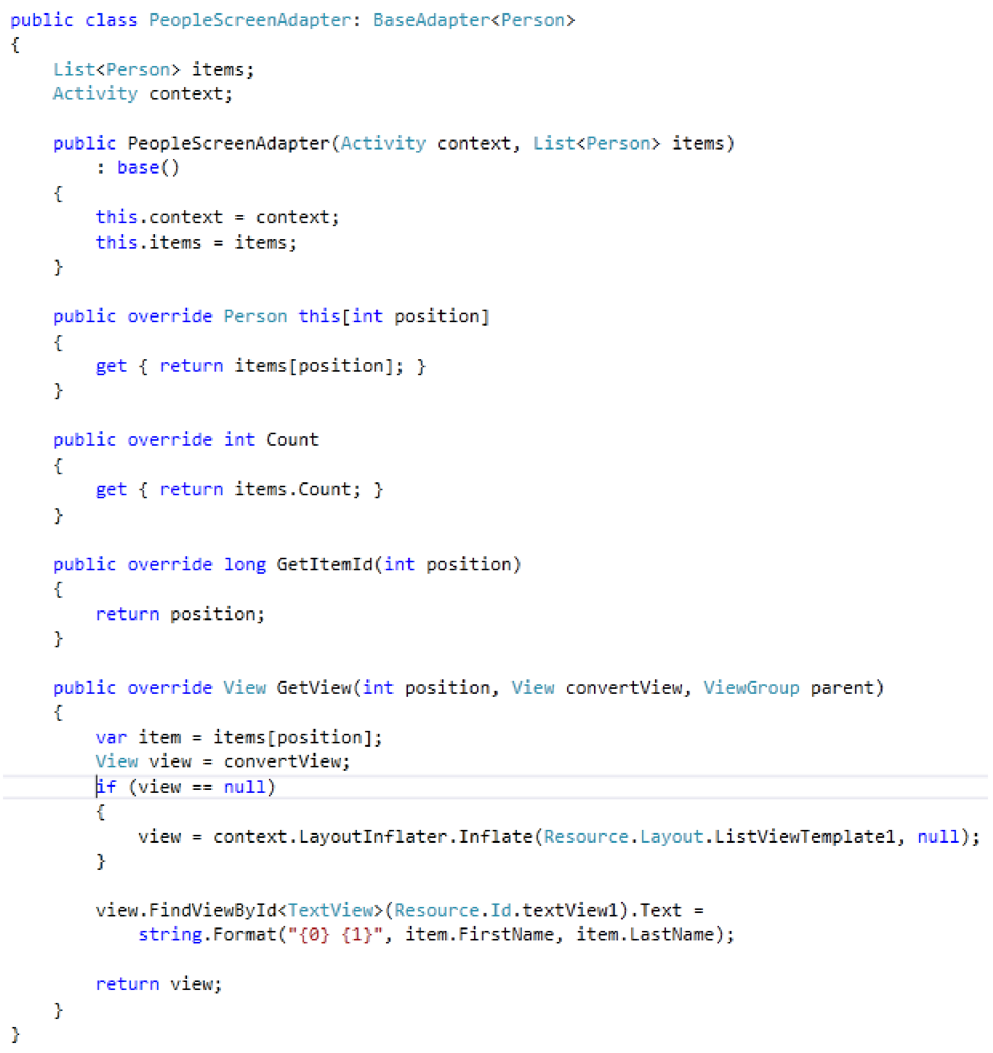
```
DisplayingEntityCollectionSample
DisplayingEntityCollectionSa
mple.Model.Person
DisplayingEntityCollectionSa
mple.Model.Person
DisplayingEntityCollectionSa
mple.Model.Person
DisplayingEntityCollectionSa
mple.Model.Person
DisplayingEntityCollectionSa
mple.Model.Person
DisplayingEntityCollectionSa
mple.Model.Person
```

Not so difficult, just create another class which would be derived from the abstract class BaseAdapter. The following properties and methods are implemented once you inherit the class.



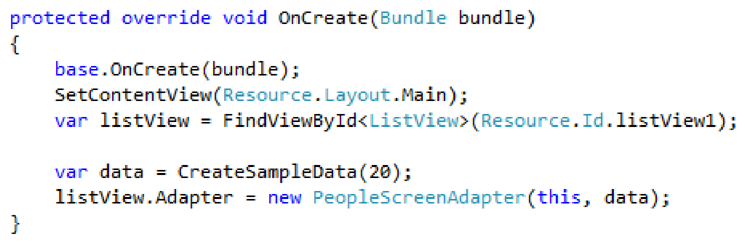
```
public class PeopleScreenAdapter: BaseAdapter<Person>
{
    public override Person this[int position]
    {
        get { throw new NotImplementedException(); }
    }

    public override int Count
    {
        get { throw new NotImplementedException(); }
    }

    public override long GetItemId(int position)
    {
        throw new NotImplementedException();
    }

    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        throw new NotImplementedException();
    }
}
```

The following modifications are done to the class, which is self-explanatory.



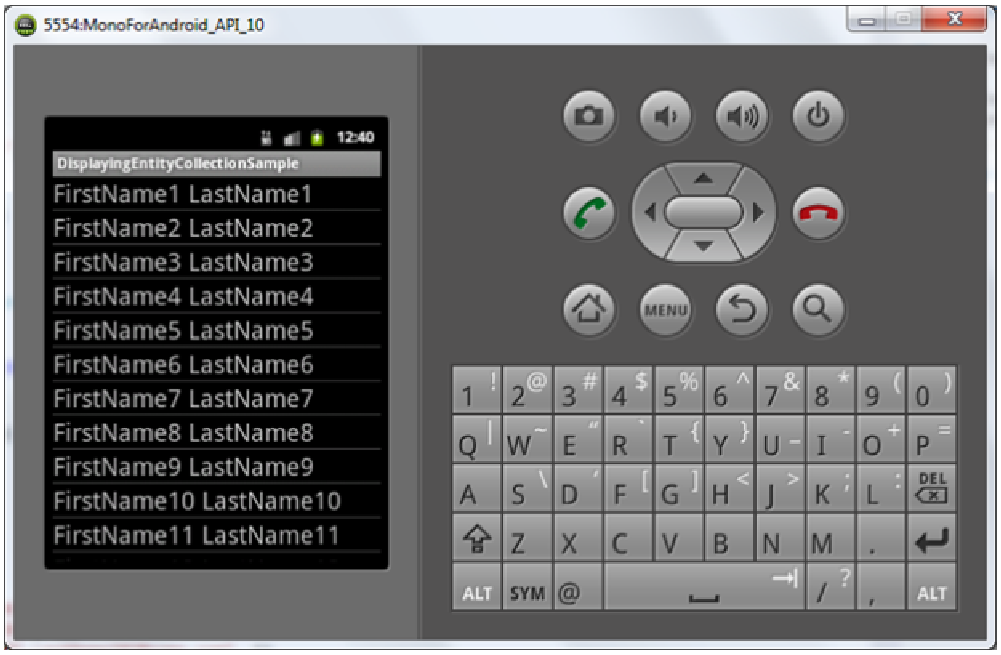
```
public class PeopleScreenAdapter: BaseAdapter<Person>
{
    List<Person> items;
    Activity context;

    public PeopleScreenAdapter(Activity context, List<Person> items)
        : base()
    {
        this.context = context;
        this.items = items;
    }

    public override Person this[int position]
    {
        get { return items[position]; }
    }

    public override int Count
    {
        get { return items.Count; }
    }

    public override long GetItemId(int position)
    {
        return position;
    }

    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        var item = items[position];
        View view = convertView;
        if (view == null)
        {
            view = context.LayoutInflater.Inflate(Resource.Layout.ListViewTemplate1, null);
        }

        view.FindViewById<TextView>(Resource.Id.textView1).Text =
            string.Format("{0} {1}", item.FirstName, item.LastName);

        return view;
    }
}
```

And back to Activity1, and do the following for displaying data in ListView.



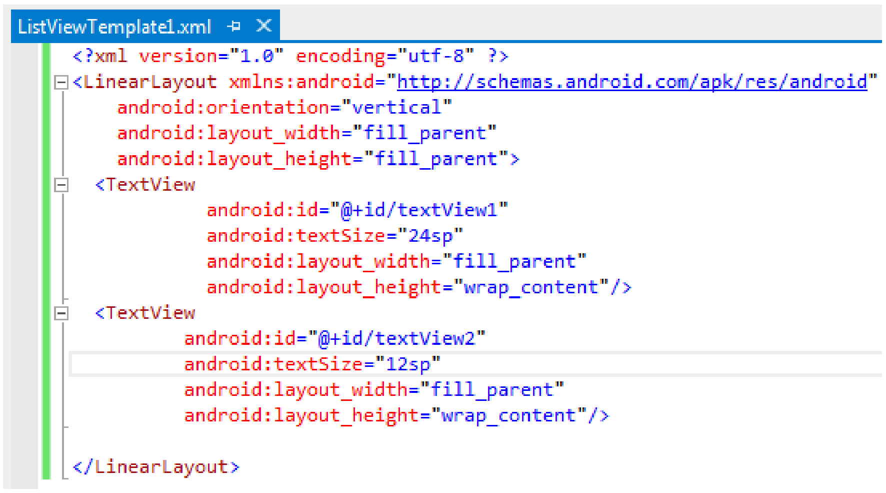
```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);
    var listView = FindViewById<ListView>(Resource.Id.listView1);

    var data = CreateSampleData(20);
    listView.Adapter = new PeopleScreenAdapter(this, data);
}
```

Well, it has displayed with a single TextView inside the template.



```
FirstName1 LastName1
FirstName2 LastName2
FirstName3 LastName3
FirstName4 LastName4
FirstName5 LastName5
FirstName6 LastName6
FirstName7 LastName7
FirstName8 LastName8
FirstName9 LastName9
FirstName10 LastName10
FirstName11 LastName11
```

How about displaying Email Id below the Full Name. This will require a change in the ListViewTemplate1 change. Following are the changes.

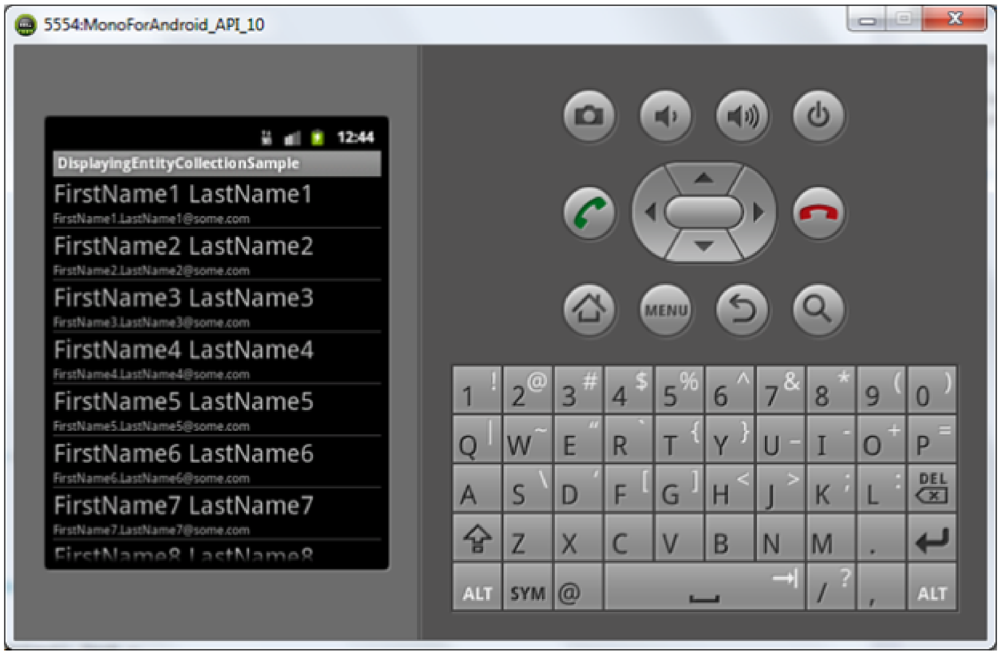


```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
  <TextView
            android:id="@+id/textView1"
            android:textSize="24sp"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
  <TextView
          android:id="@+id/textView2"
          android:textSize="12sp"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"/>

</LinearLayout>
```

And in the Screen Adapter class, GetView would be changed like this.

```
public override View GetView(int position, View convertView, ViewGroup parent)
{
    var item = items[position];
    View view = convertView;
    if (view == null)
    {
        view = context.LayoutInflater.Inflate(Resource.Layout.ListViewTemplate1, null);
    }

    view.FindViewById<TextView>(Resource.Id.textView1).Text =
        string.Format("{0} {1}", item.FirstName, item.LastName);
    view.FindViewById<TextView>(Resource.Id.textView2).Text = item.EmailId;

    return view;
}
```

Now run the application, and you would see the Email Id is also displayed.


5554:MonoForAndroid_API_10

12:44

DisplayingEntityCollectionSample

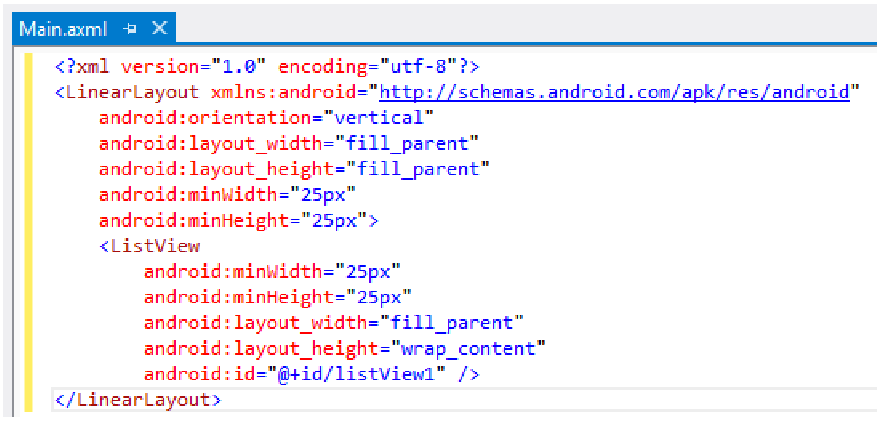
FirstName1 LastName1
FirstName1.LastName1@some.com
FirstName2 LastName2
FirstName2.LastName2@some.com
FirstName3 LastName3
FirstName3.LastName3@some.com
FirstName4 LastName4
FirstName4.LastName4@some.com
FirstName5 LastName5
FirstName5.LastName5@some.com
FirstName6 LastName6
FirstName6.LastName6@some.com
FirstName7 LastName7
FirstName7.LastName7@some.com

As you see a in the above output, two TextView controls are displayed.

## Selective ListView

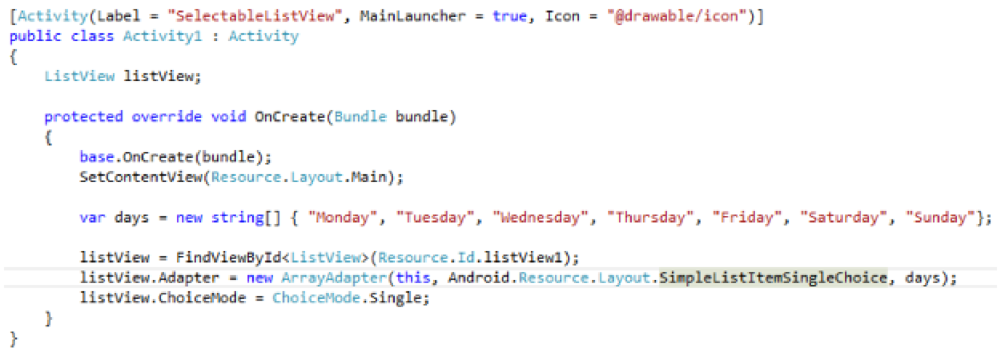
Here we would see how to display a collection in ListView and select one or many items.

As we are going to have a ListView, the following is the simple layout.



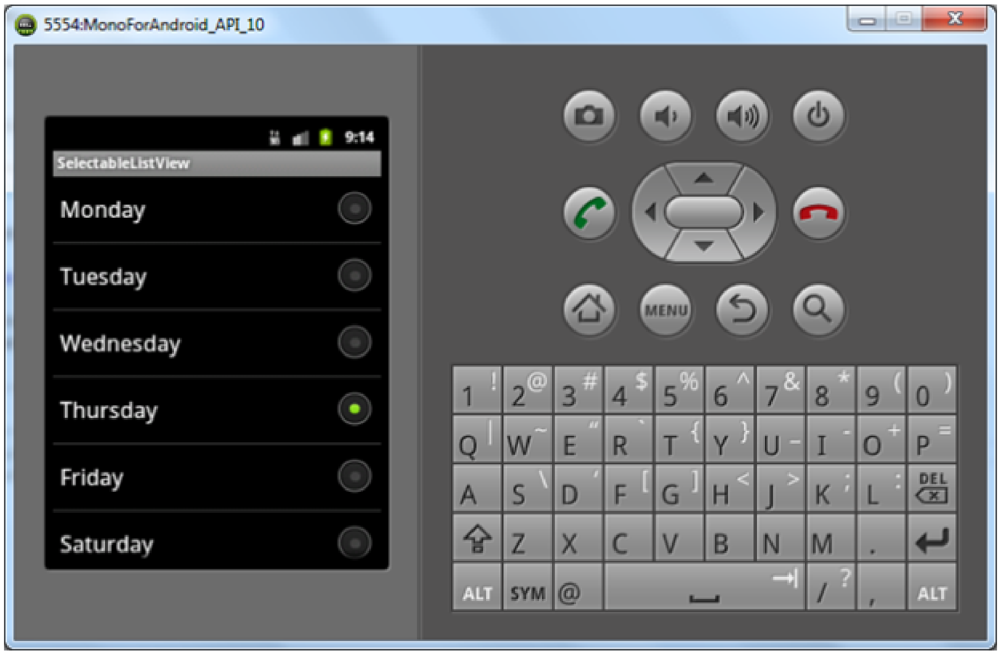
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:minWidth="25px"
    android:minHeight="25px">
    <ListView
        android:minWidth="25px"
        android:minHeight="25px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/listView1" />
</LinearLayout>
```

Now instead of creating a layout for the ListView content, we would use some from the Android.Resource . And adding the ChoiceMode property to the ListView.

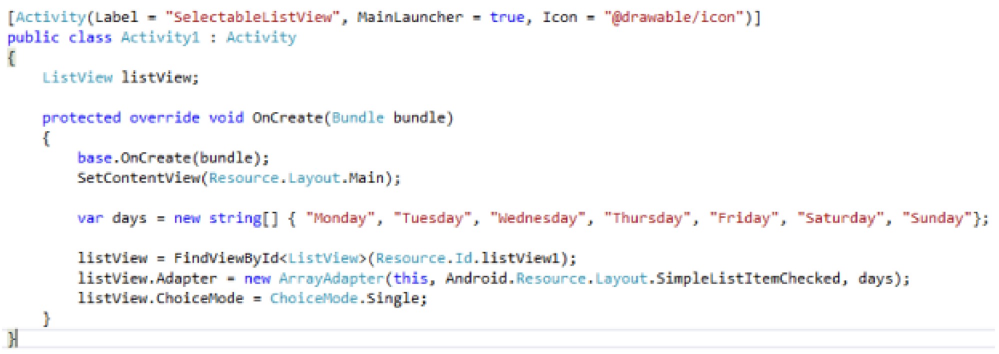


```csharp
[Activity(Label = "SelectableListView", MainLauncher = true, Icon = "@drawable/icon")]
public class Activity1 : Activity
{
    ListView listView;

    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.Main);

        var days = new string[] { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

        listView = FindViewById<ListView>(Resource.Id.listView1);
        listView.Adapter = new ArrayAdapter(this, Android.Resource.Layout.SimpleListItemSingleChoice, days);
        listView.ChoiceMode = ChoiceMode.Single;
    }
}
```

Here is the view, with single selection mode with Radio Buttons.

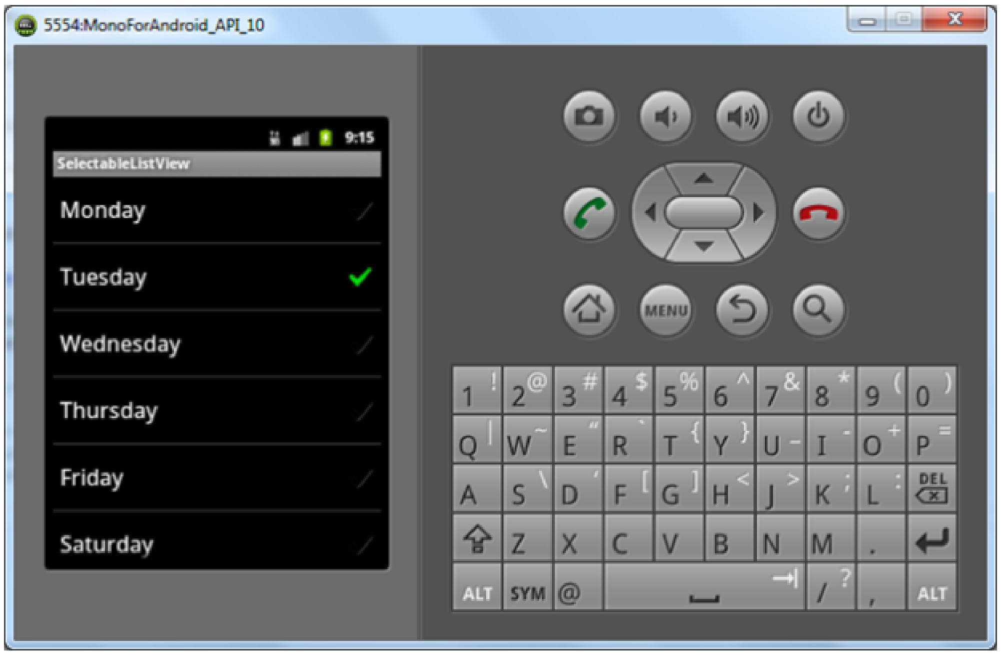


We have a CheckBox style instead of RadioButton, here are the changes.

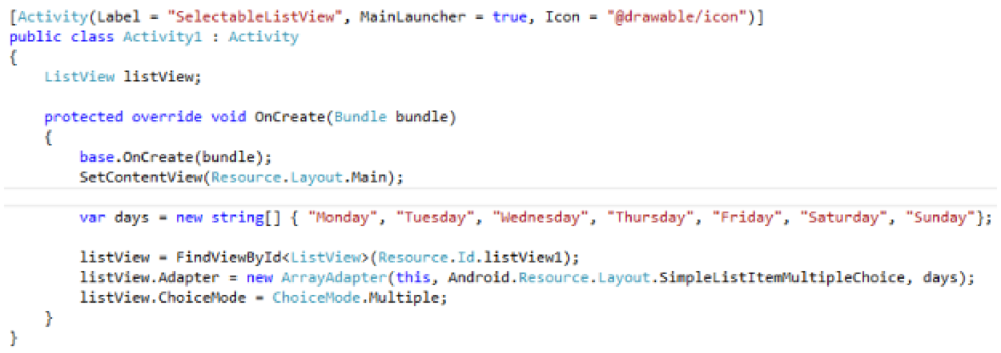


```
[Activity(Label = "SelectableListView", MainLauncher = true, Icon = "@drawable/icon")]
public class Activity1 : Activity
{
    ListView listView;

    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.Main);

        var days = new string[] { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

        listView = FindViewById<ListView>(Resource.Id.listView1);
        listView.Adapter = new ArrayAdapter(this, Android.Resource.Layout.SimpleListItemChecked, days);
        listView.ChoiceMode = ChoiceMode.Single;
    }
}
```

And you would get the Check image instead of the RadioButton.

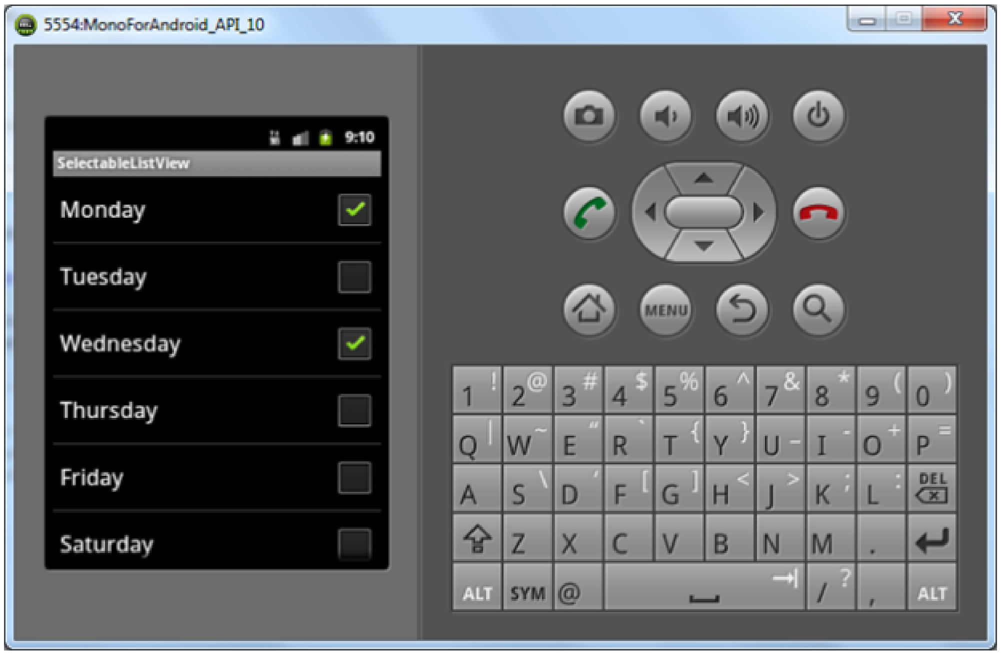


Now, let’s try for the multiple selections. Also you need to change the ChoiceMode to Multiple.



```
[Activity(Label = "SelectableListView", MainLauncher = true, Icon = "@drawable/icon")]
public class Activity1 : Activity
{
    ListView listView;

    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.Main);

        var days = new string[] { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};

        listView = FindViewById<ListView>(Resource.Id.listView1);
        listView.Adapter = new ArrayAdapter(this, Android.Resource.Layout.SimpleListItemMultipleChoice, days);
        listView.ChoiceMode = ChoiceMode.Multiple;
    }
}
```

And you would get the multiple checked items.

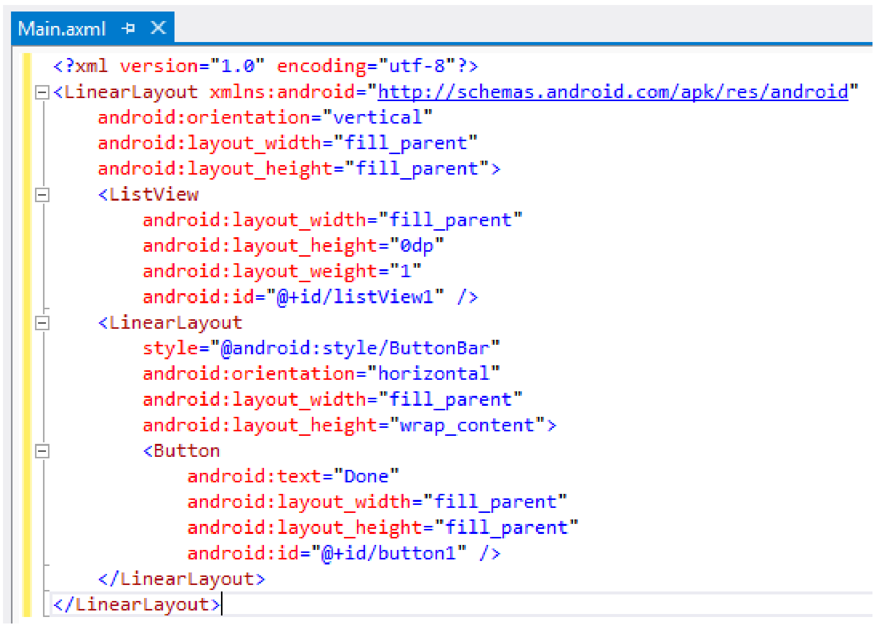


The choice modes are clearly useful for having single or multiple choice interactions from user.

## Get Selected Items from ListView

Previously we had seen to display a list where user could select single or multiple. Here we would see, how to get the selected items from the screen activity where a list of options are displayed.

The following is the layout for displaying the Multiple Selectable ListView with a button.

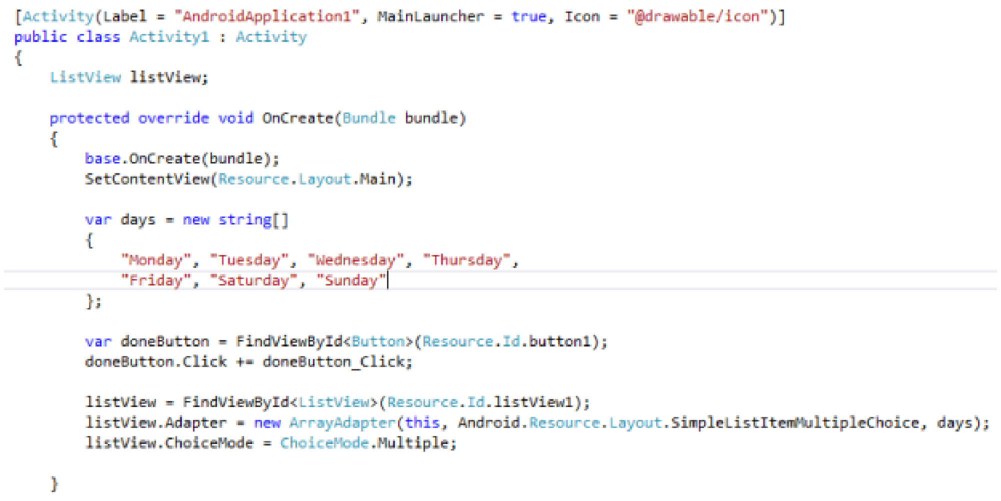


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ListView
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:id="@+id/listView1" />
    <LinearLayout
        style="@android:style/ButtonBar"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <Button
            android:text="Done"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/button1" />
    </LinearLayout>
</LinearLayout>
```

Now we would bind data to the ListView.

Also we would make it Multiple Selectable with SelectOption as Multiple.

And we would subscribe to the click event of the Button.



```
[Activity(Label = "AndroidApplication1", MainLauncher = true, Icon = "@drawable/icon")]
public class Activity1 : Activity
{
    ListView listView;

    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.Main);

        var days = new string[]
        {
            "Monday", "Tuesday", "Wednesday", "Thursday",
            "Friday", "Saturday", "Sunday"
        };

        var doneButton = FindViewById<Button>(Resource.Id.button1);
        doneButton.Click += doneButton_Click;

        listView = FindViewById<ListView>(Resource.Id.listView1);
        listView.Adapter = new ArrayAdapter(this, Android.Resource.Layout.SimpleListItemMultipleChoice, days);
        listView.ChoiceMode = ChoiceMode.Multiple;

    }
```

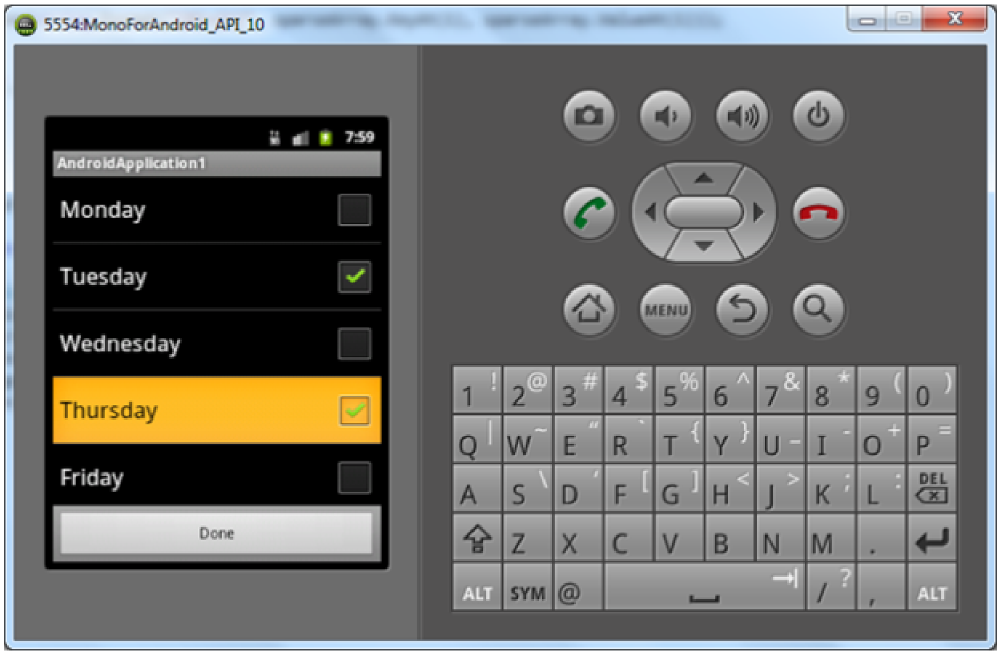
In the button click event handler, we would do as following to get the selected data out of ListView.

```
void doneButton_Click(object sender, EventArgs e)
{
    var builder = new StringBuilder();
    var sparseArray = FindViewById<ListView>(Resource.Id.listView1).CheckedItemPositions;


    for (var i = 0; i < sparseArray.Size(); i++)
    {
        builder.AppendLine(string.Format("{0}={1}", sparseArray.KeyAt(i), sparseArray.ValueAt(i)));
    }

    ShowAlert("ListView", builder.ToString());
}
```

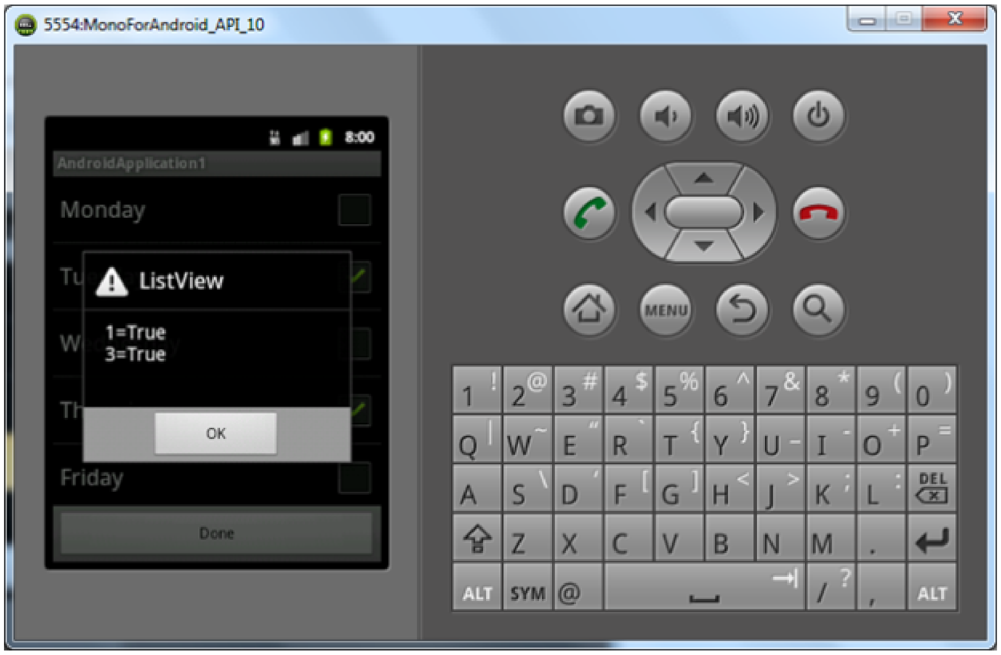
ShowAlert is a helper method to show alerts with easy, here is the code for that too.

```
private void ShowAlert(string title, string message)
{
    Android.App.AlertDialog.Builder builder = new AlertDialog.Builder(this);
    AlertDialog alertDialog = builder.Create();
    alertDialog.SetTitle(title);
    alertDialog.SetIcon(Android.Resource.Drawable.IcDialogAlert);
    alertDialog.SetMessage(message);
    alertDialog.SetButton("OK", (s, ev) =>
    {
        //DO Something
    });
    alertDialog.Show();
}
```

Now run the application. You could see the fixed.



As on button click, the selected items positions with respect to the collection is displayed.



ListView

1=True
3=True

The selected items are displayed in the alert dialog, but you might find out that, even if a selection is made and reverted back; the item's state was registered as false or true. In that case we need to check based on the true values in the collection.

## ListView in a ScrollView Issue

In Windows Phone application development, use of ListBox and ScrollViewer have no issues in scrolling the content. But in Android using both are not advisable. They might behave differently.

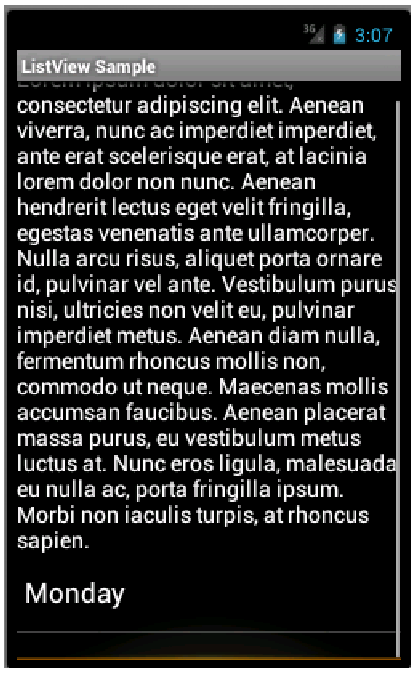
Here we would explore that, and find out how we try to fix this.

```
<ScrollView xmlns:p1="http://schemas.android.com/apk/res/android"
    p1:minWidth="25px"
    p1:minHeight="25px"
    p1:layout_width="fill_parent"
    p1:layout_height="fill_parent"
    p1:id="@+id/scrollView1">
    <LinearLayout
        p1:orientation="vertical"
        p1:minWidth="25px"
        p1:minHeight="25px"
        p1:layout_width="fill_parent"
        p1:layout_height="fill_parent"
        p1:id="@+id/linearLayout1">
        <TextView
            p1:text="@string/sample_text"
            p1:textAppearance="?android:attr/textAppearanceMedium"
            p1:layout_width="fill_parent"
            p1:layout_height="wrap_content"
            p1:id="@+id/textView1" />
        <ListView
            p1:minWidth="25px"
            p1:minHeight="25px"
            p1:layout_width="fill_parent"
            p1:layout_height="wrap_content"
            p1:id="@+id/listView1" />
    </LinearLayout>
</ScrollView>
```

The above layout consists of a ScrollView as the parent ViewGroup, which has two elements in it. A TextView and a ListView. For the issue to happen, we would have some long text in the TextView.

```
ListView listView1;

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);
    listView1 = FindViewById<ListView>(Resource.Id.listView1);

    var days = new string[]
    {
        "Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday", "Sunday"
    };

    listView1.Adapter = new ArrayAdapter<string>
            (this, Android.Resource.Layout.SimpleListItem1, days);
}
```

Now we would bind data to the Adapter of the ListView so that data would be displayed. Now let’s run the application.



consectetur adipiscing elit. Aenean viverra, nunc ac imperdiet imperdiet, ante erat scelerisque erat, at lacinia lorem dolor non nunc. Aenean hendrerit lectus eget velit fringilla, egestas venenatis ante ullamcorper. Nulla arcu risus, aliquet porta ornare id, pulvinar vel ante. Vestibulum purus nisi, ultricies non velit eu, pulvinar imperdiet metus. Aenean diam nulla, fermentum rhoncus mollis non, commodo ut neque. Maecenas mollis accumsan faucibus. Aenean placerat massa purus, eu vestibulum metus luctus at. Nunc eros ligula, malesuada eu nulla ac, porta fringilla ipsum. Morbi non iaculis turpis, at rhoncus sapien.

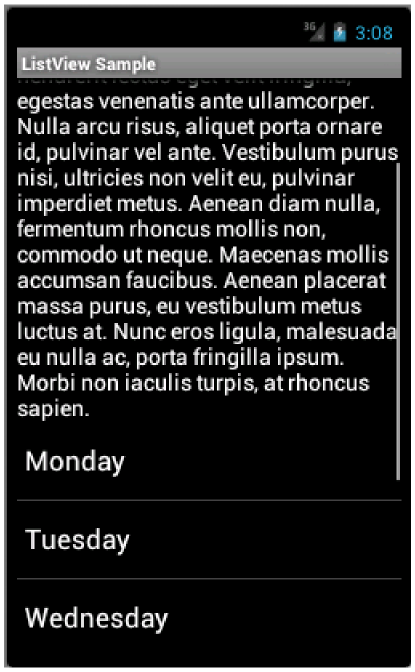
As you see in the above output, the TextView is scrolled to the end, and the ListVIew is partially displayed. To confirm that, you also notice that, we have a scrollbar scrolled to the end of the view. Also the bottom reached orange display is displayed. Enough confirmation, now to fix that we need a method which does the following.

```
public static class ControlHelper
{
    public static void FixListViewSize(ListView listView)
    {
        IListAdapter adapter = listView.Adapter;

        if (adapter == null)
        {
            return;
        }

        var totalHeight = 0;
        for (int i = 0; i < adapter.Count; i++)
        {
            View listItem = adapter.GetView(i, null, listView);
            listItem.Measure(0, 0);
            totalHeight += listItem.MeasuredHeight;
        }

        ViewGroup.LayoutParams parameters = listView.LayoutParameters;
        parameters.Height = totalHeight +
                    (listView.DividerHeight * (adapter.Count - 1));

        listView.LayoutParameters = parameters;
    }
}
```

As you see in the above method, individually takes the item from the adapter and adds to the total. And finally it assigns to the height of the ViewGroup.

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);
    listView1 = FindViewById<ListView>(Resource.Id.listView1);

    var days = new string[]
    {
        "Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday", "Sunday"
    };

    listView1.Adapter = new ArrayAdapter<string>
            (this, Android.Resource.Layout.SimpleListItem1, days);

    ControlHelper.FixListViewSize(listView1);
}
```

To use the above method, we need to call it, just after you assign adapter of the ListView.



egestas venenatis ante ullamcorper. Nulla arcu risus, aliquet porta ornare id, pulvinar vel ante. Vestibulum purus nisi, ultricies non velit eu, pulvinar imperdiet metus. Aenean diam nulla, fermentum rhoncus mollis non, commodo ut neque. Maecenas mollis accumsan faucibus. Aenean placerat massa purus, eu vestibulum metus luctus at. Nunc eros ligula, malesuada eu nulla ac, porta fringilla ipsum. Morbi non iaculis turpis, at rhoncus sapien.

Monday

Tuesday

Wednesday

Now, the above output looks fixed. The ScrollBar is still in the middle, that means we have some items to display from the ListView.

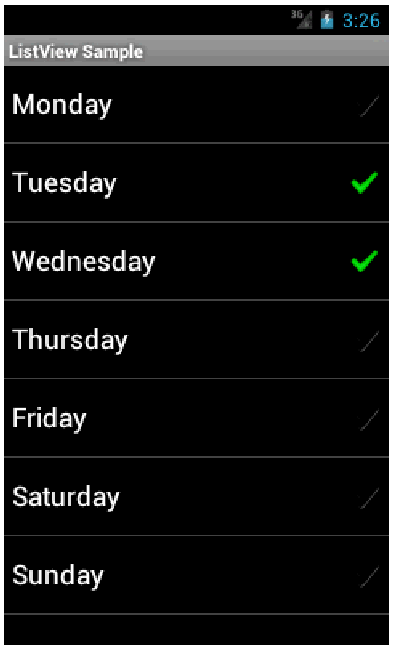
A quick note though, this has been tested with Android 4.0 and above, the results were successful.

## Selective ListActivity

In Activity chapter, we had a glance to look at the ListActivity, also in this chapter we have seen a selective ListView. Now we would combine it and provide a selective ListActivity.

```
public class Activity1 : ListActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);

        var days = new string[]
        {
            "Monday", "Tuesday", "Wednesday", "Thursday",
            "Friday", "Saturday", "Sunday"
        };

        ListAdapter = new ArrayAdapter<string>(this,
                Android.Resource.Layout.SimpleListItemChecked, days);
        ListView.ChoiceMode = ChoiceMode.Multiple;
    }
```

As you see in the above code display, we are assigning the ArrayAdapter to the ListAdapter property of the ListActivity class. It has also the property for ChoiceMode. You could assign your required choice mode and it would work like ListView.



The above output shows the implementation of selective in the ListActivity.

## Selective ListActivity with Entity

Previously, we have seen that, binding a string array to the ListAdapter and displaying data. If you have an entity collection, how do we bind?

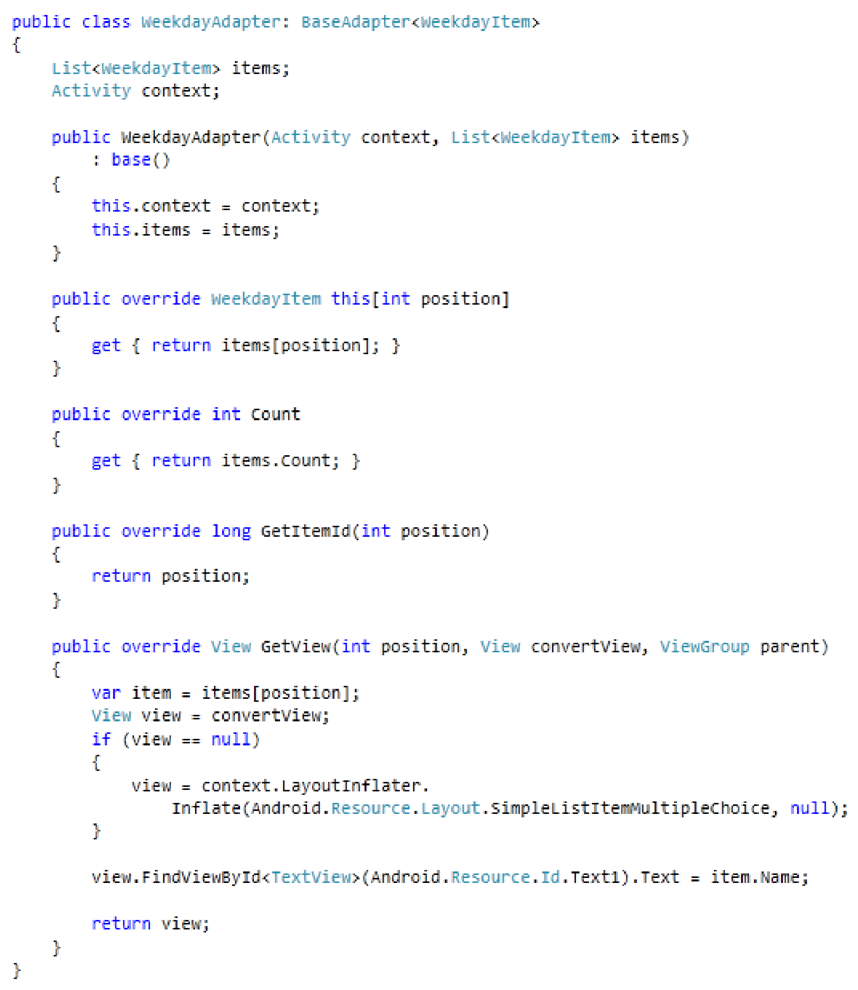
Consider WeekdayItem is the entity.

```
public class WeekdayItem
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

As you see in the above class, it has two properties Id and Name.

```
public class Activity1 : ListActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);

        var weekdays = new List<WeekdayItem>
        {
            new WeekdayItem{Id=1, Name="Monday"},
            new WeekdayItem{Id=2, Name="Tuesday"},
            new WeekdayItem{Id=3, Name="Wednesday"},
            new WeekdayItem{Id=4, Name="Thursday"},
            new WeekdayItem{Id=5, Name="Friday"},
            new WeekdayItem{Id=6, Name="Saturday"},
            new WeekdayItem{Id=7, Name="Sunday"}
        };


        ListAdapter = new WeekdayAdapter(this, weekdays);
        ListView.ChoiceMode = ChoiceMode.Multiple;
    }
```

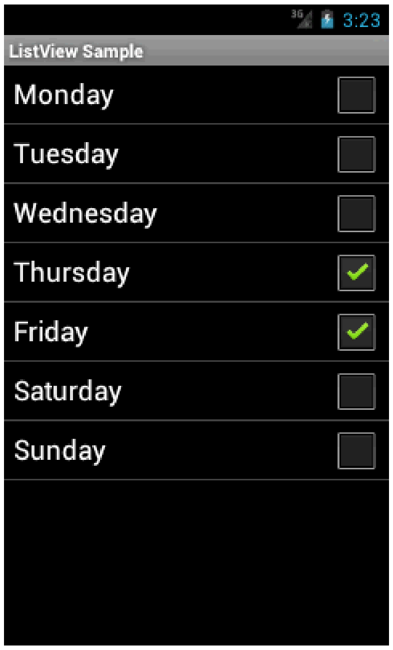
Now we would have a sample collection of WeekdayItem as List



```
public class WeekdayAdapter: BaseAdapter<WeekdayItem>
{
    List<WeekdayItem> items;
    Activity context;

    public WeekdayAdapter(Activity context, List<WeekdayItem> items)
        : base()
    {
        this.context = context;
        this.items = items;
    }

    public override WeekdayItem this[int position]
    {
        get { return items[position]; }
    }

    public override int Count
    {
        get { return items.Count; }
    }

    public override long GetItemId(int position)
    {
        return position;
    }

    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        var item = items[position];
        View view = convertView;
        if (view == null)
        {
            view = context.LayoutInflater.
                Inflate(Android.Resource.Layout.SimpleListItemMultipleChoice, null);
        }

        view.FindViewById<TextView>(Android.Resource.Id.Text1).Text = item.Name;

        return view;
    }
}
```

It is derived from BaseAdapter abstract class. Now in the GetView overridden method, we are finding the TextView and displaying the day name.

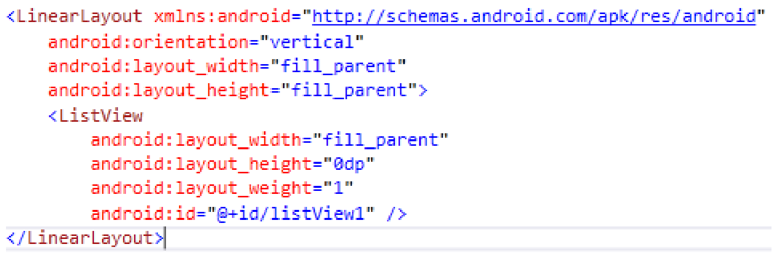
Notice that we have used an Android Resource Layout. And to find out the TextView inside it we are using the Resource Id Text1.



In the above output, the entity collection is successfully displayed in the ListActivity.

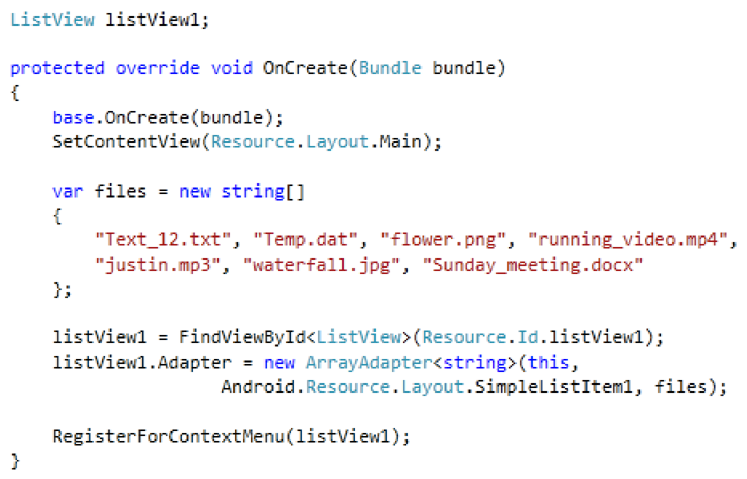
## Context Menu in ListView Item

Here, we would see how we add context menu to the ListView Item. Let's have the layout like below.



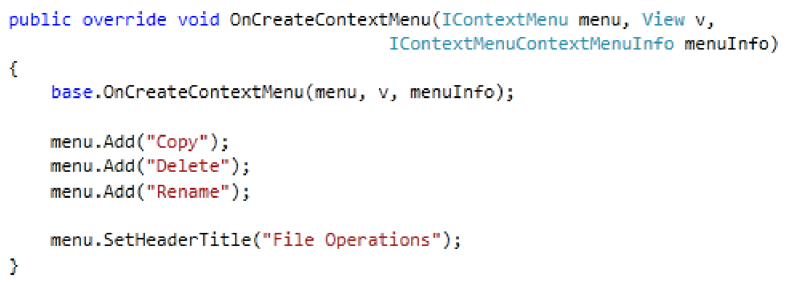
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ListView
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:id="@+id/listView1" />
</LinearLayout>
```

In the above layout, we have taken ListView to display the data.



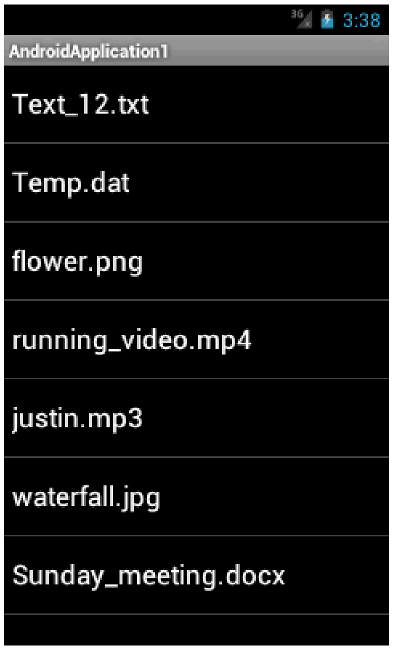
```
ListView listView1;

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    var files = new string[]
    {
        "Text_12.txt", "Temp.dat", "flower.png", "running_video.mp4",
        "justin.mp3", "waterfall.jpg", "Sunday_meeting.docx"
    };

    listView1 = FindViewById<ListView>(Resource.Id.listView1);
    listView1.Adapter = new ArrayAdapter<string>(this,
                    Android.Resource.Layout.SimpleListItem1, files);

    RegisterForContextMenu(listView1);
}
```

Now, let's have a file collection and display it in the ListView. Also we would call RegisterForContextMenu method which takes the ListView.

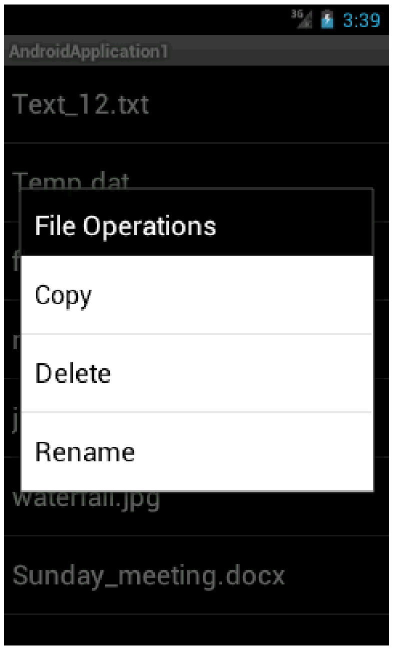


```
public override void OnCreateContextMenu(IContextMenu menu, View v,
                                        IContextMenuContextMenuInfo menuInfo)
{
    base.OnCreateContextMenu(menu, v, menuInfo);

    menu.Add("Copy");
    menu.Add("Delete");
    menu.Add("Rename");

    menu.SetHeaderTitle("File Operations");
}
```

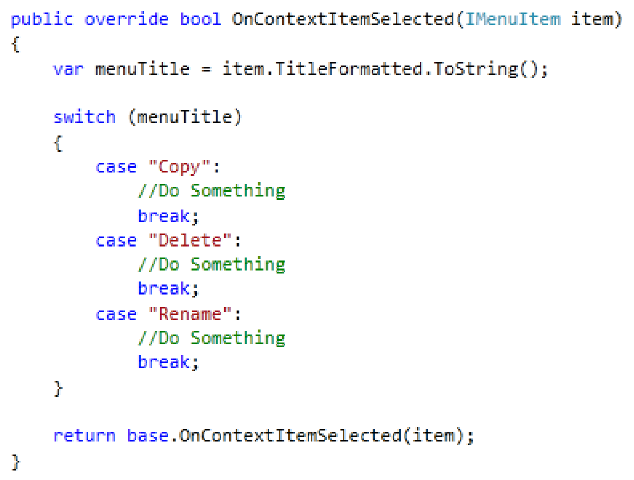
We need to override the OnCreateContextMenu method and add the menu items and also the title of the context menu. As we are having the files displayed, let's have a menu for file operations.



Now, the above output is for displaying file names in the ListView.

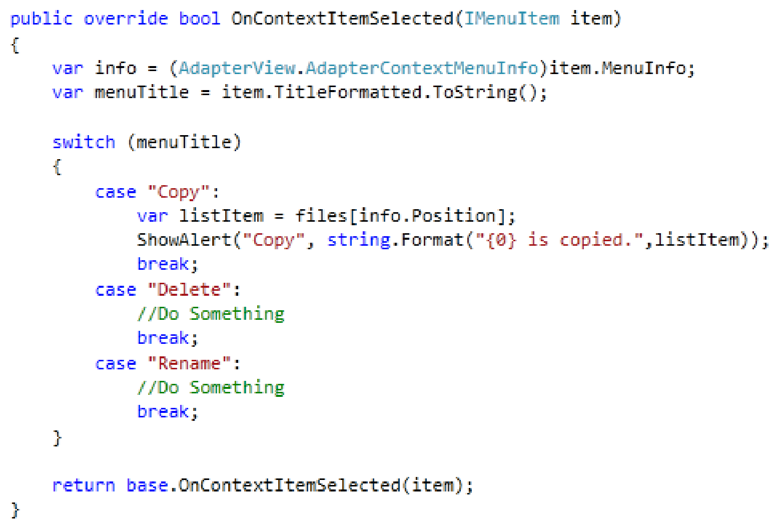


Now, Tap and Hold; then the context menu would be displayed like above.



```
public override bool OnContextItemSelected(IMenuItem item)
{
    var menuTitle = item.TitleFormatted.ToString();

    switch (menuTitle)
    {
        case "Copy":
            //Do Something
            break;
        case "Delete":
            //Do Something
            break;
        case "Rename":
            //Do Something
            break;
    }

    return base.OnContextItemSelected(item);
}
```

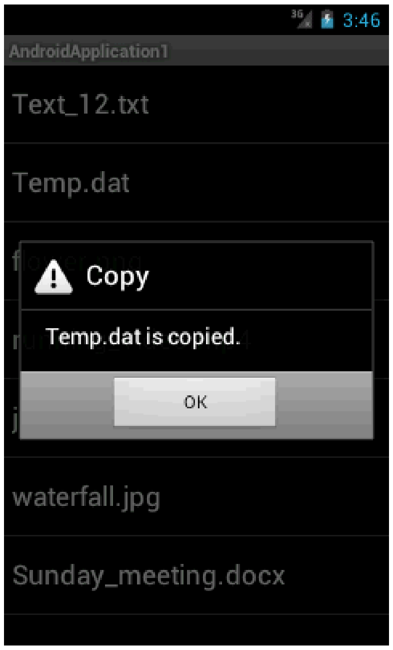
To know which context menu item you have selected; you could use the above code to find it out.



```
public override bool OnContextItemSelected(IMenuItem item)
{
    var info = (AdapterView.AdapterContextMenuInfo)item.MenuInfo;
    var menuTitle = item.TitleFormatted.ToString();

    switch (menuTitle)
    {
        case "Copy":
            var listItem = files[info.Position];
            ShowAlert("Copy", string.Format("{0} is copied.",listItem));
            break;
        case "Delete":
            //Do Something
            break;
        case "Rename":
            //Do Something
            break;
    }

    return base.OnContextItemSelected(item);
}
```

### Get ListView Item with Context menu selection

Also to operate on the ListView item on context menu selection is required.



Temp.dat is copied.

In the above code display, the IMenuItem type is type casted to AdapterView.AdapterContextMenuInfo. This would give us the position of the item in the Adapter. Which we could find out in our code.

## Displaying selective Image in ListView

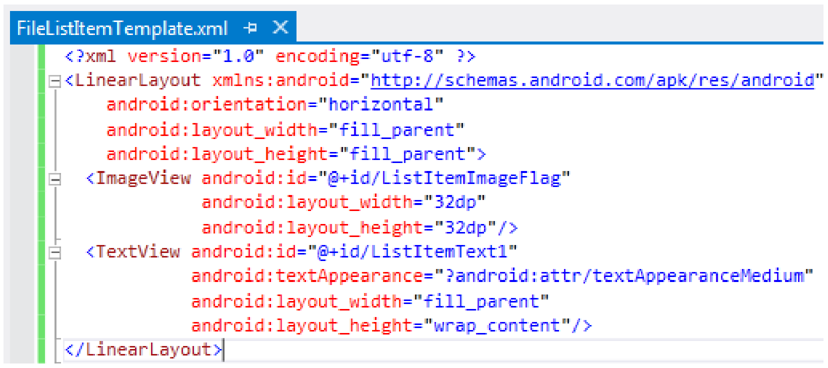
The above output shows that, the Temp.data file was tapped to display the context menu. And the copy menu was selected from the context menu. The dialog box confirms that.

```
public class FileItem
{
    public int Id { get; set; }
    public string Name { get; set; }
    public bool IsUpdated { get; set; }
}
```

In some scenarios, you need to display a image (change the image source) based on some value. For example stock prices go up, then up arrow; and when coming down, down arrow image. Here we would follow up on our previous example.

```
<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:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:id="@+id/listView1" />
</LinearLayout>
```

As you see in the above FileItem entity class a bool IsUpdated property is present. As the name suggests it determines whether the file is synced or not.



FileListItemTemplate.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
  <ImageView android:id="@+id/ListItemImageFlag"
             android:layout_width="32dp"
             android:layout_height="32dp"/>
  <TextView android:id="@+id/ListItemText1"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
</LinearLayout>
```

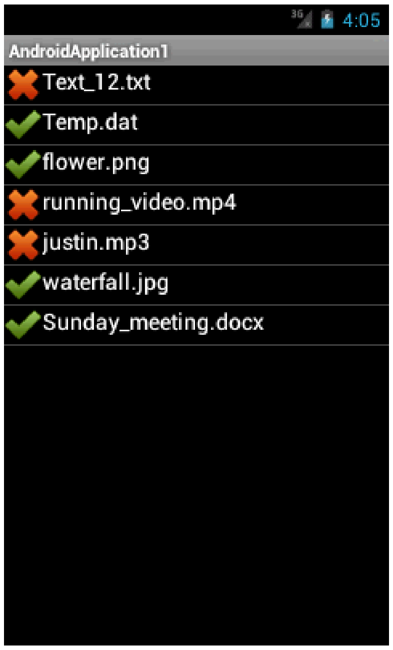
Let’s have the above ListView layout. Also we require an item template for the ListView.

```
public override View GetView(int position, View convertView, ViewGroup parent)
{
    var item = items[position];
    View view = convertView;
    if (view == null)
    {
        view = context.LayoutInflater.
            Inflate(Resource.Layout.FileListItemTemplate, null);
    }

    view.FindViewById<TextView>(Resource.Id.ListItemText1).Text = item.Name;

    var image = view.FindViewById<ImageView>(Resource.Id.ListItemImageFlag);

    if (item.IsUpdated)
    {
        image.SetImageResource(Resource.Drawable.correct_icon);
    }
    else
    {
        image.SetImageResource(Resource.Drawable.wrong_icon);
    }

    return view;
}
```

The above layout shows that, we have an ImageView and a TextView in the template.

```
ListView listView1;
List<FileItem> files;

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    files = new List<FileItem>
    {
        new FileItem{Name="Text_12.txt", IsUpdated=false},
        new FileItem{Name="Temp.dat", IsUpdated=true},
        new FileItem{Name="flower.png", IsUpdated=true},
        new FileItem{Name="running_video.mp4", IsUpdated=false},
        new FileItem{Name="justin.mp3", IsUpdated=false},
        new FileItem{Name="waterfall.jpg", IsUpdated=true},
        new FileItem{Name="Sunday_meeting.docx", IsUpdated=true}
    };

    listView1 = FindViewById<ListView>(Resource.Id.listView1);
    listView1.Adapter = new FileListAdapter(this, files);
}
```

In the above GetView method from the custom adapter class, finding out the TextView and assigns the text to it. Also the ImageView's image is changed. Assume that you have added two different images for the flag display. The Resource is set by using the SetImageResource method.



```
Text_12.txt
Temp.dat
flower.png
running_video.mp4
justin.mp3
waterfall.jpg
Sunday_meeting.docx
```

The above output consists of the selective image display based on the condition check.

## Conclusion

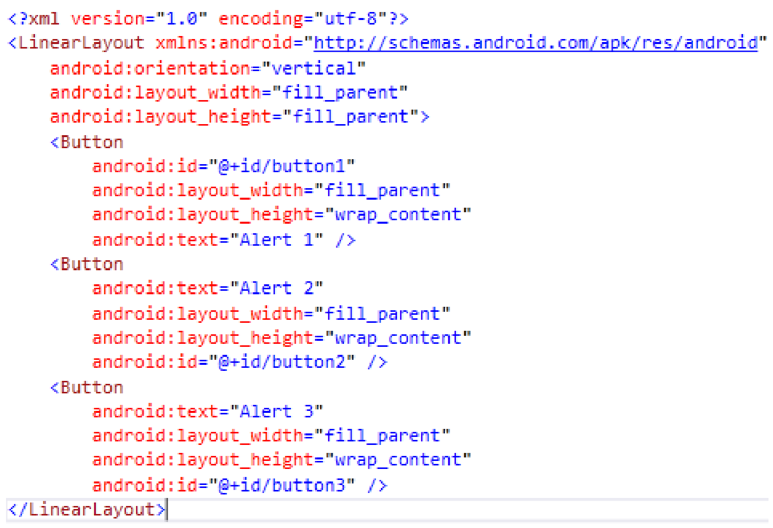
In this chapter we have seen ListVIew specifically. Different scenarios with ListView were presented with their solution approach.

# Dialogs and Toast

A dialog is a small window that is prompted to the user to make a decision or enter additional information. We have 3 types of dialogs defined in Android system.

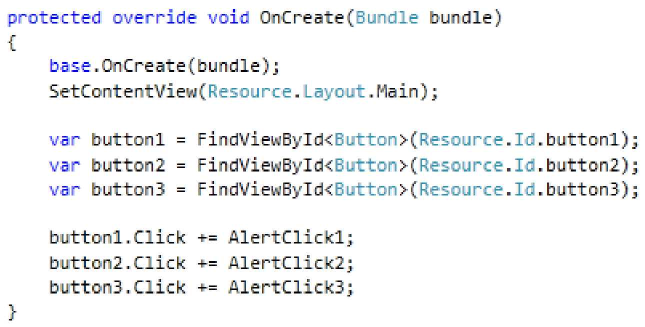
## Alert Dialog

This is a type of dialog, where content is displayed with a dialog title and a button. The following layout is for the example of displaying alert dialogs. We have three buttons.



```
<?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">
    <Button
        android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Alert 1" />
    <Button
        android:text="Alert 2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/button2" />
    <Button
        android:text="Alert 3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/button3" />
</LinearLayout>
```

The following code is very simple; we are getting the controls from the layout and subscribing to the respective click events.



```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    var button1 = FindViewById<Button>(Resource.Id.button1);
    var button2 = FindViewById<Button>(Resource.Id.button2);
    var button3 = FindViewById<Button>(Resource.Id.button3);

    button1.Click += AlertClick1;
    button2.Click += AlertClick2;
    button3.Click += AlertClick3;
}
```

### Single Button Alert Dialog

In the following example the basic single button dialog is displayed.

```
void AlertClick1(object sender, EventArgs e)
{
    Android.App.AlertDialog.Builder builder = new AlertDialog.Builder(this);
    AlertDialog alertDialog = builder.Create();
    alertDialog.SetTitle("Alert Title");
    alertDialog.SetIcon(Android.Resource.Drawable.IcDialogAlert);
    alertDialog.SetMessage("This is a sample message.");
    alertDialog.SetButton("OK", (s, ev) =>
    {
        //DO Something
    });
    alertDialog.Show();
}
```

As you see in the above code snippet, we are creating a builder. The builder is assigned to AlertDialog object. Then we are setting Title, Icon, and Message. Also we are setting the button OK. A single button is mandatory for displaying dialogs. Now, subscribing to the click event if you need to do something here.

### Two Buttons Alert Dialog

The following code snippet is for creating two buttons in Alert Dialog. As you see in the below code snippet; we are setting buttons using SetButton and SetButton2. This is same as first button; the second button is considered here as Negative and the first as positive.

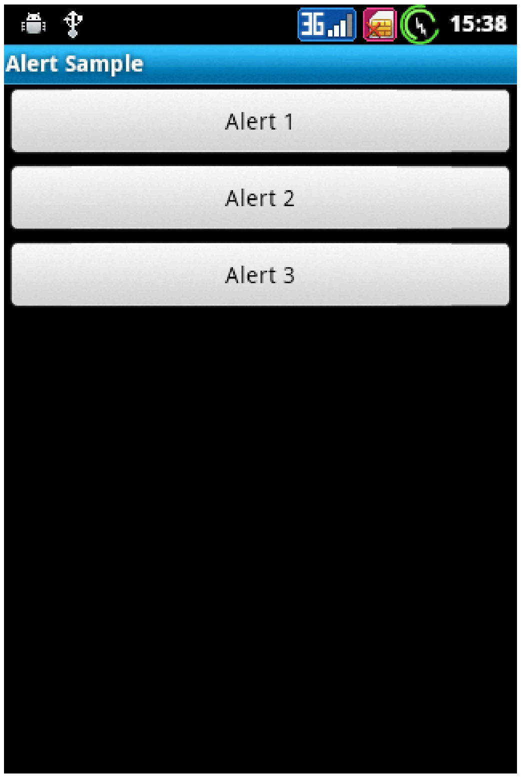
```
void AlertClick2(object sender, EventArgs e)
{
    Android.App.AlertDialog.Builder builder = new AlertDialog.Builder(this);

    AlertDialog alertDialog = builder.Create();
    alertDialog.SetTitle("Alert Title");
    alertDialog.SetIcon(Resource.Drawable.Icon);
    alertDialog.SetMessage("This is a sample message.");

    //YES
    alertDialog.SetButton("Yes", (s, ev) =>
    {
        //DO Something
    });

    //NO
    alertDialog.SetButton2("No", (s, ev) =>
    {
        //DO Something
    });

    alertDialog.Show();
}
```

### Three Buttons Alert Dialog

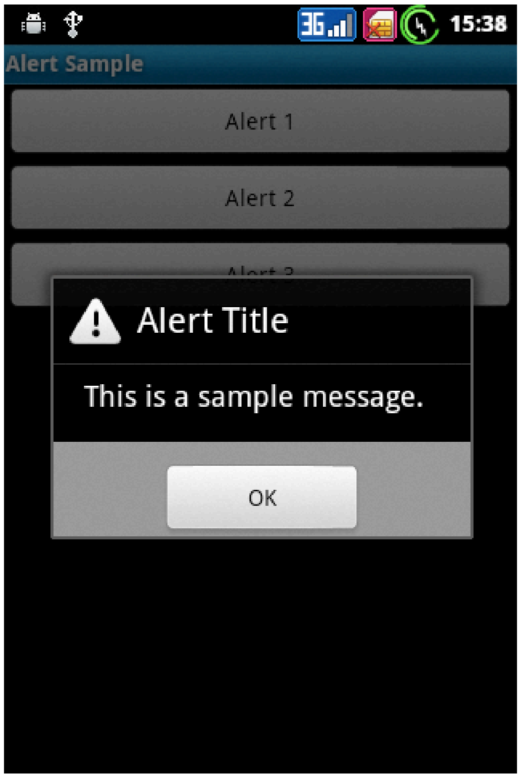
The following code snippet would help you understand creating three buttons in Alert Dialog. From previous code, we have added another SetButton3. Here the SetButton is considered as positive, SetButton3 as negative and the SetButton2 as neutral.

```
void AlertClick3(object sender, EventArgs e)
{
    Android.App.AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.SetCancelable(true);
    AlertDialog alertDialog = builder.Create();
    alertDialog.SetTitle("Alert Title");
    alertDialog.SetIcon(Resource.Drawable.Icon);
    alertDialog.SetMessage("This is a sample message.");

    //YES
    alertDialog.SetButton("Yes", (s, ev) =>
    {
        //DO Something
    });

    //Cancel
    alertDialog.SetButton2("Cancel", (s, ev) =>
    {
        //DO Something
    });

    //No
    alertDialog.SetButton3("No", (s, ev) =>
    {
        //DO Something
    });

    alertDialog.Show();
}
```

Let’s run the application and you would see the first screen where all three buttons are there.

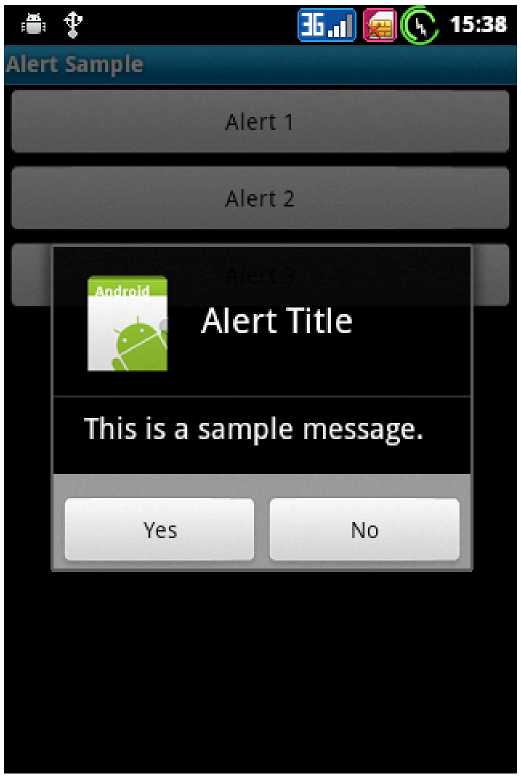


Let’s click the first button.



A simple alert dialog with a Title, Message and a button is displayed.

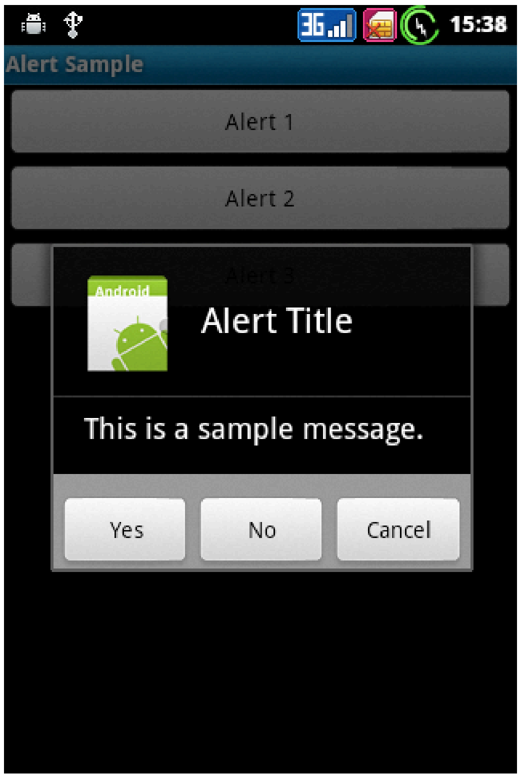
Now clicking the second button.



This is a sample message.

An icon is displayed with the title; also we have two buttons displayed. This dialog could be used for confirmations.

And finally we would click on the third button.



Three buttons are displayed here; this is where you would take user's interaction properly.

## Date Picker Dialog

We have a DatePickerDialog, which would help to pick a date. The following example would help you how we use it.

```
private const int DATE_DIALOG_ID = 0;
DateTime dateTime;
TextView textView1;

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    var button1 = FindViewById<Button>(Resource.Id.button1);
    textView1 = FindViewById<TextView>(Resource.Id.textView1);

    dateTime = DateTime.Now;
    textView1.Text = string.Format("{0}-{1}-{2}",
                        dateTime.Year, dateTime.Month, dateTime.Day);

    button1.Click += ButtonClick1;
}

void ButtonClick1(object sender, EventArgs e)
{
    ShowDialog(DATE_DIALOG_ID);
}
```

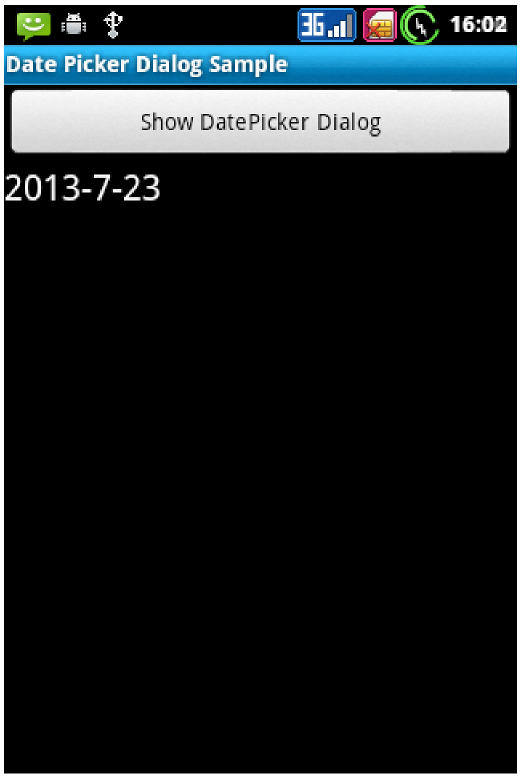
As you see in the above code, we are declaring a DateTime, a TextView and an Int variable. Then the button's click event is subscribed. In the button click event handler we are calling the method ShowDialog which takes the int value. The int value is here to distinguish a dialog from each other; in case you have more than one dialog.

```
protected override Dialog OnCreateDialog(int id)
{
    switch (id)
    {
        case DATE_DIALOG_ID:
            var dialog = new DatePickerDialog(this, DatePickerDataSet,
                            dateTime.Year, dateTime.Month - 1, dateTime.Day);
            return dialog;
    }

    return null;
}

private void DatePickerDataSet(object sender,
                                    DatePickerDialog.DateSetEventArgs e)
{
    this.dateTime = e.Date;
    textView1.Text = string.Format("{0}-{1}-{2}",
                        dateTime.Year, dateTime.Month, dateTime.Day);
}
```

In the above code snippet; we are overriding OnCreateDialog method from the Activity class. It takes the integer Id as parameter. As we discussed previously there could be multiple dialogs in a single activity; so we would be using a switch case scenario. Also we are creating a new dialog of type DatePickerDialog. The DatePickerDialog takes the context, the event handler on set, the year, the month and the day as parameters.

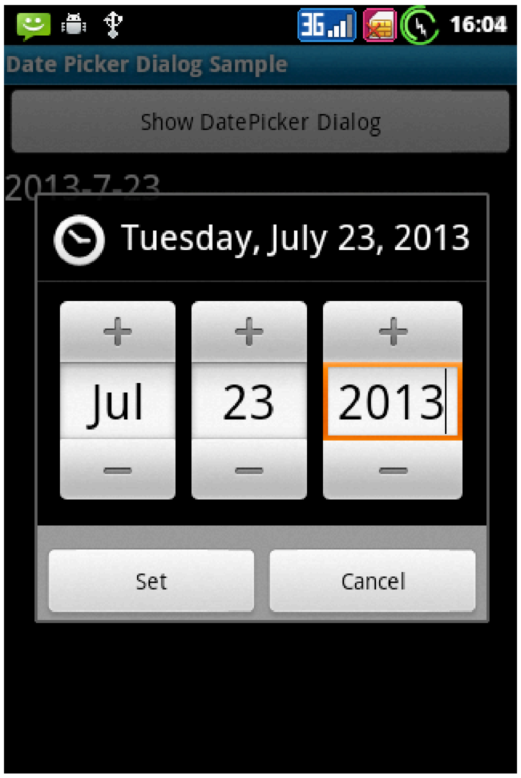
You might notice that, the month is subtracted by 1; this is because you don't subscribe it by 1, the next month would be displayed. It might be from Android itself.

We would run the application and click on the button.

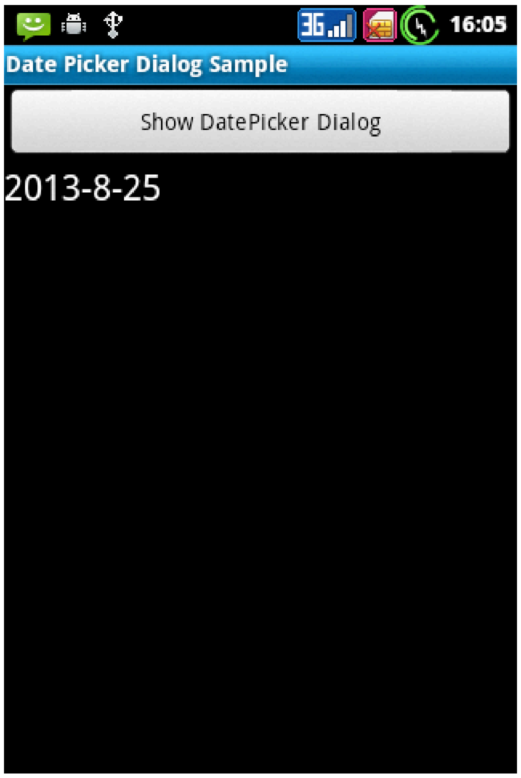


2013-7-23

The date picker dialog is displayed. You could select the date and click on set to set the date and click on cancel to cancel this operation.



On set click the date is set to the text view.

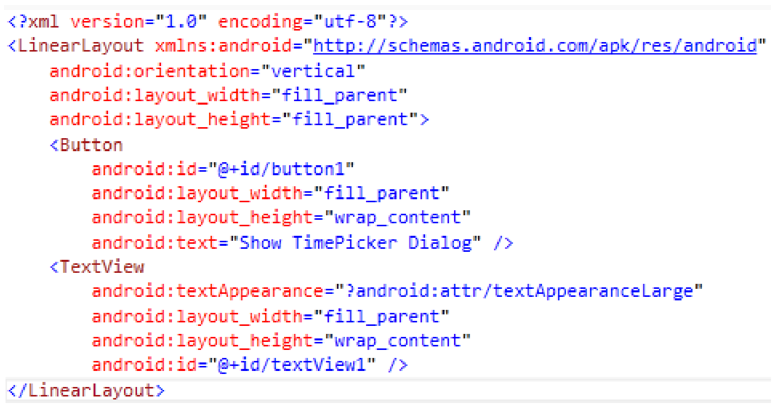


2013-8-25

## Time Picker Dialog

TimePickerDialog is almost similar in using as using DatePickerDialog. The following example would help you using it.

As you see below; we have a button and a TextView in the layout. The button is to bring up the dialog and text view is where the value would be set.



```
<?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">
    <Button
        android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Show TimePicker Dialog" />
    <TextView
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView1" />
</LinearLayout>
```

Now similar to the Previous DatePickerDialog example, we would have an integer variable for Id, an integer variable for hour and one for minute, then TextView variable. Instead of taking hour and minute as integer variables; you could take a TimeSpan variable and use the Hour and Minute properties from it.

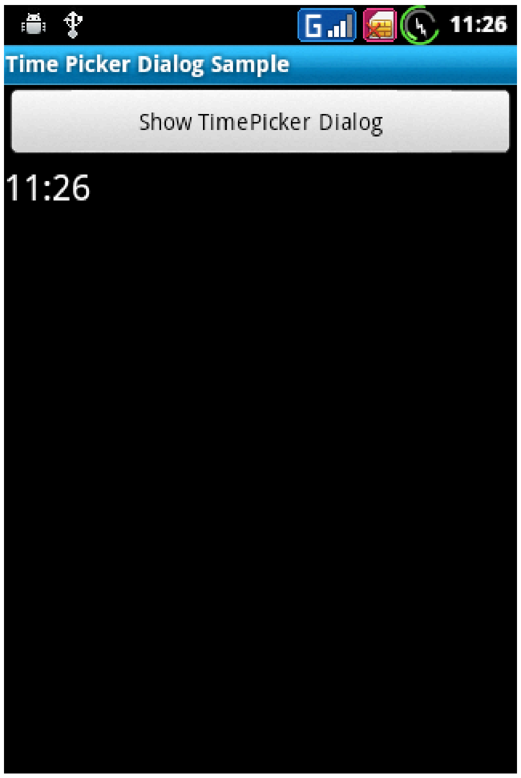
```
private const int TIME_DIALOG_ID = 0;
int hour;
int minute;
TextView textView1;

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    var button1 = FindViewById<Button>(Resource.Id.button1);
    textView1 = FindViewById<TextView>(Resource.Id.textView1);

    hour = DateTime.Now.Hour;
    minute = DateTime.Now.Minute;

    textView1.Text = string.Format("{0}:{1}", hour, minute);

    button1.Click += ButtonClick1;
}

void ButtonClick1(object sender, EventArgs e)
{
    ShowDialog(TIME_DIALOG_ID);
}
```

As you see in the above code, the hour and minute are set to current hour and minute values. Also in the button's click event we are calling the method ShowDialog, to show the dialog with the id.

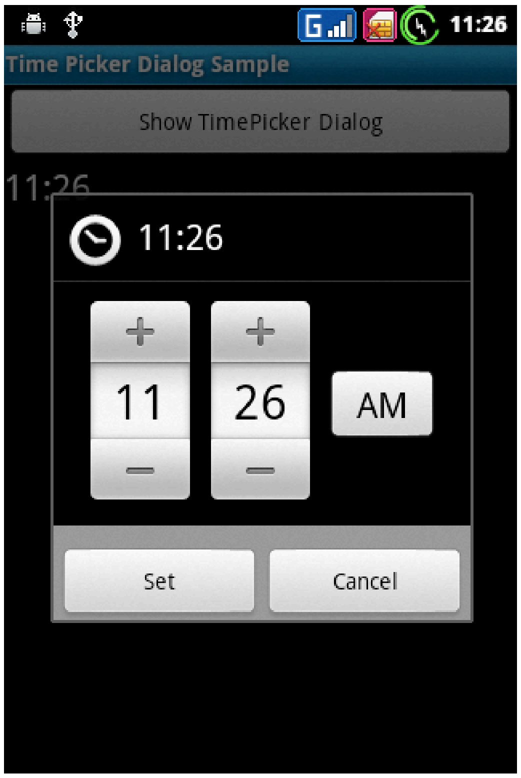
```
protected override Dialog OnCreateDialog(int id)
{
    switch (id)
    {
        case TIME_DIALOG_ID:
            var dialog = new TimePickerDialog(this,
                          TimePickerDataSet,hour, minute, false);
            return dialog;
    }

    return null;
}

private void TimePickerDataSet(object sender,
                                    TimePickerDialog.TimeSetEventArgs e)
{
    hour = e.HourOfDay;
    minute = e.Minute;
    textView1.Text = string.Format("{0}:{1}", hour, minute);
}
```

As we discussed previously, we are having the overridden method OnCreateDialog from Activity class where Id is the parameter for identifying dialogs in the activity. And on TimePickerDataSet handler we are setting the selected values.

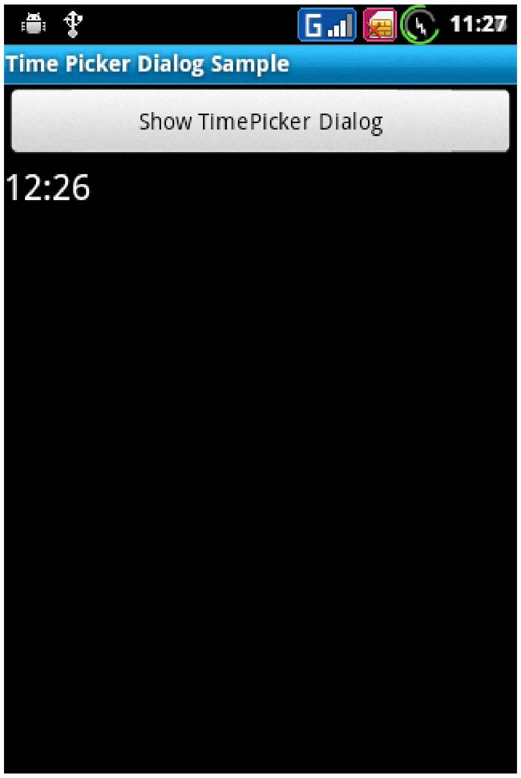
Let's run the application.



On click of the button, the following Timer Picker dialog is displayed. You notice that the AM/PM changer is also available. This is because we had set the final parameter in creating the TimePickerDialog object. The final parameter is for identifying the 24 hour format or 12 hour format.



On setting the time the value is set in the handler; where the values are taken from the TimeSetEventArgs.



## Toast

A toast is a simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

We have seen dialogs which take user interaction, but in case of toasts user interaction is not needed. For example: you have created some task successfully and you want to notify the user that it's a success.

The following example would help you to use it.

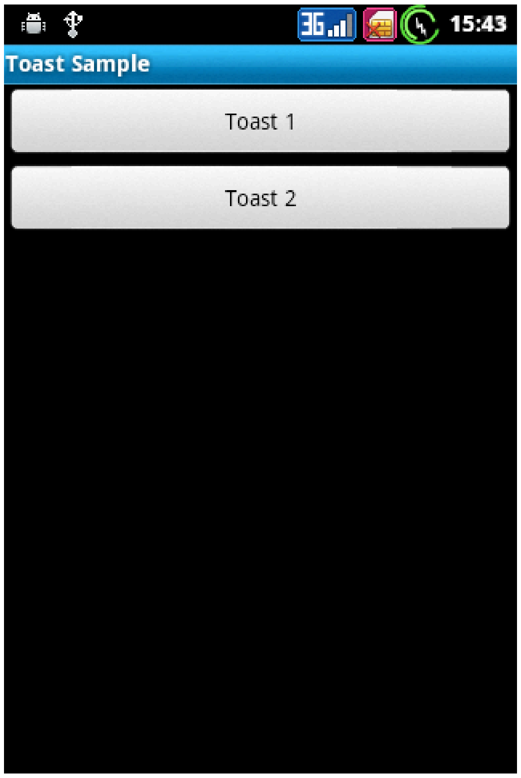
```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    SetContentView(Resource.Layout.Main);

    var button1 = FindViewById<Button>(Resource.Id.button1);
    var button2 = FindViewById<Button>(Resource.Id.button2);

    button1.Click += ToastClick1;
    button2.Click += ToastClick2;
}
```

As you see in the above code snippet; familiar code is written where two buttons are extracted from the layout and the respective click events are subscribed.

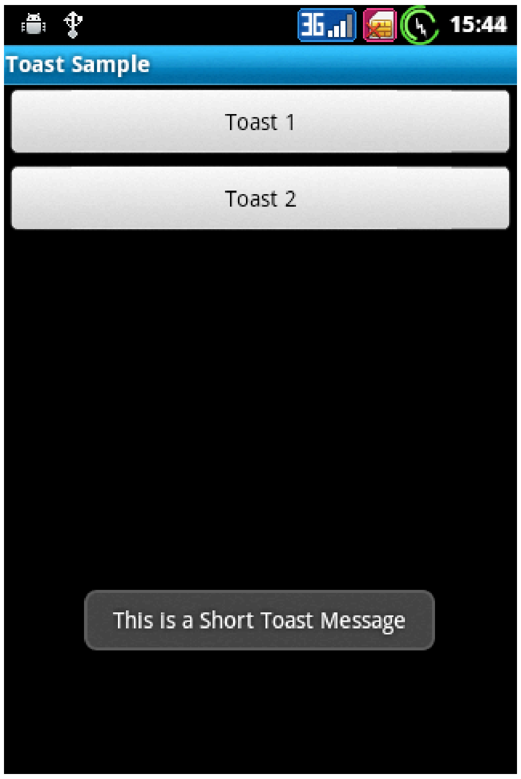
```
void ToastClick1(object sender, EventArgs e)
{
    var toast = Toast.MakeText(this,
                "This is a Short Toast Message", ToastLength.Short);
    toast.Show();
}

void ToastClick2(object sender, EventArgs e)
{
    var toast = Toast.MakeText(this,
                  "This is a Long Toast Message", ToastLength.Long);
    toast.Show();
}
```

As you see in the above code snippet; two toasts are displayed; the first toast is of short length and the second is longer in length.

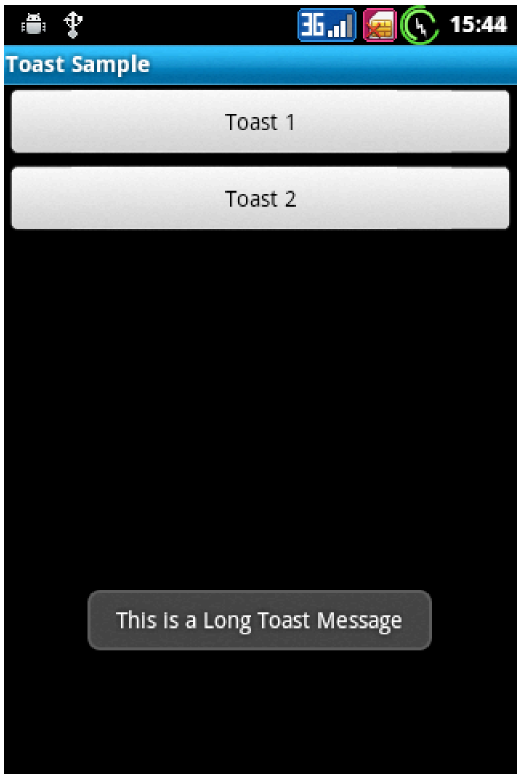
Let's run the application.



On click of the specific buttons the short and long toast is displayed.



This is a Short Toast Message



This is a Long Toast Message

## Conclusion

In this chapter we have seen that dialogs are used for getting some interaction from the user. The simple alert dialog could be used for displaying messages. Toasts should be used in case you are doing some operation and user waits whether it's a success or failure.