## Cross-Platform Mobile Application Development with Xamarin: Module 3, Lesson 5

## Build a Basic Android app using Xamarin.Android Lab Overview

Xamarin is a powerful tool that allows you to develop mobile applications across all major mobile platforms. This lab will cover how to build a very basic Android application using Xamarin.Android. In this lab TextViews, Buttons, ListViews, Spinners and other objects and concepts will be explored.

Objectives

In this hands-on lab you will learn how to:

* Interact with Xamarin, specifically in the context of building an Android app
* Create a text input field
* Display a dialog box on the click of a button
* Create and display a spinner that displays a list of names

Prerequisites

The following are required to complete this hands-on lab:

* Visual Studio with Xamarin installed or Xamarin Studio for OS X
* Completion of [Module 3 Lessons 1-5](https://github.com/MSFTImagine/computerscience/tree/master/Instructor-Led/Module3/Lessons)

Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Create a First Name Entry UI
* Exercise 2: Create a Spinner of Names
* Exercise 3: Create a ListView

Exercise 1: Create a First Name Entry UI

This exercise covers creating a first name entry UI that displays the entered name in a Toast when a button is clicked.

1. Create a Visual Studio Android Blank App Project
2. Using the Android Designer, in Resources/layout/Main.axml, delete the default button
   * Add a Text (Large) TextView with a with a text value of “First Name”.

<TextView

android:text="First Name"

android:textAppearance="?android:attr/textAppearanceLarge"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/name\_text" />

* + Add an Plain Text (EditText) control called first\_name with no text value.

<EditText

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/first\_name"/>

* + Add a Button called submit\_name.

<Button

android:text="Button"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/submit\_name" />

1. In MainActivity.cs, make the button click event show the firstName value in a toast.

First, locate the views on the layout.

Button button = FindViewById<Button>(Resource.Id.submit\_name);

EditText editText = FindViewById<EditText>(Resource.Id.first\_name);

Next, handle the button’s Click event.

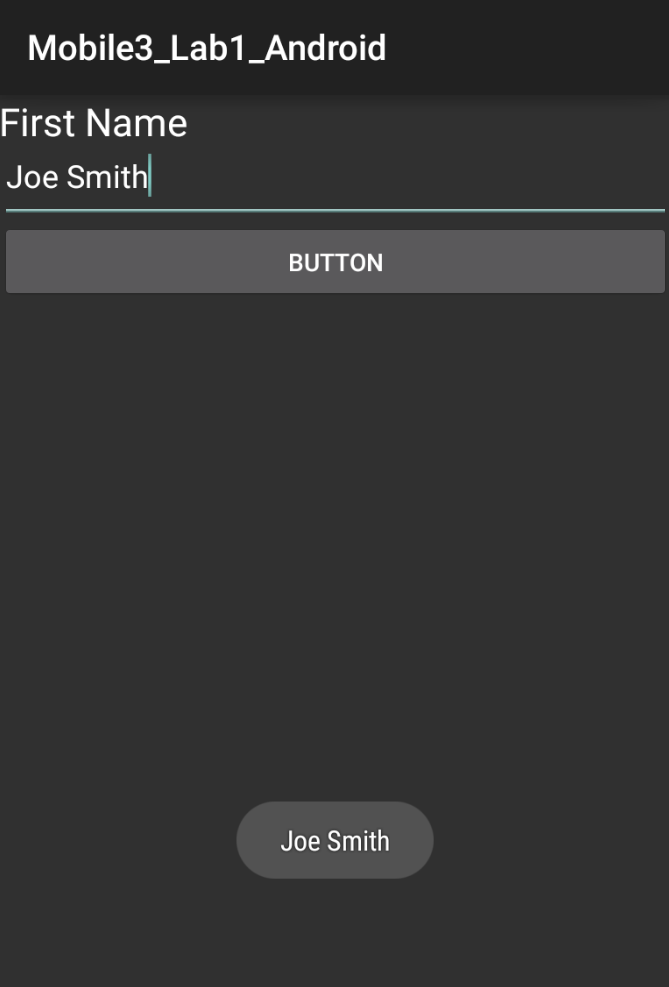
button.Click += delegate { … }

OR

button.Click += (object sender, EventArgs e) => {…}

In the button’s event, create and show the toast.

Toast.MakeText(this, editText.Text, ToastLength.Long).Show();



Tip: The previous example was a simple EditText situation. Real-life examples might also use the EditText TextChanged or KeyPress events.

## Exercise 2: Create a Spinner of Names

Create spinner for selection from a list of names in an array of strings. When a name is selected, display the name in a toast.

1. Create a new Android layout (Project > Add New Item) called spinner\_list.axml and using the Android Designer:
   * Add a TextView called fullNameLabel with Text value “Name”.
   * Add a Spinner called fullNameSpinner.
2. Create a new layout called text\_view\_for\_spinner.axml that contains only a TextView. Remove the LinearLayout because this layout is for spinner rows only. Remember to add an xmlns attribute.

<TextView xmlns:android="http://schemas.android.com/apk/res/android"

android:text="Medium Text"

android:textAppearance="?android:attr/textAppearanceMedium"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/textView1" />

1. Create a new activity called SpinnerList.cs. Make the fullNameSpinner display a list of names of your choice. When one is selected display it in a toast.

Set this activity as the startup activity by setting MainLauncher to true.

[Activity(Label = "SpinnerList", MainLauncher = true)]

Remember to set the MainLauncher property in the Main Activity to false.

In the SpinnerList class (public class SpinnerList : Activity {…}) be sure to create a spinner named spinner

Spinner spinner;

In the SpinnerList activity’s OnCreate method, load the text\_view\_for\_spinner layout.

SetContentView(Resource.Layout.spinner\_list);

Find the spinner view on the layout.

Spinner editText = FindViewById<Spinner>(Resource.Id.fullNameSpinner);

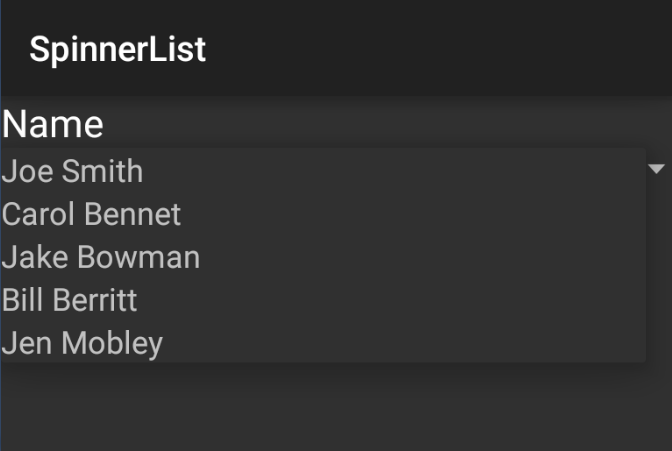
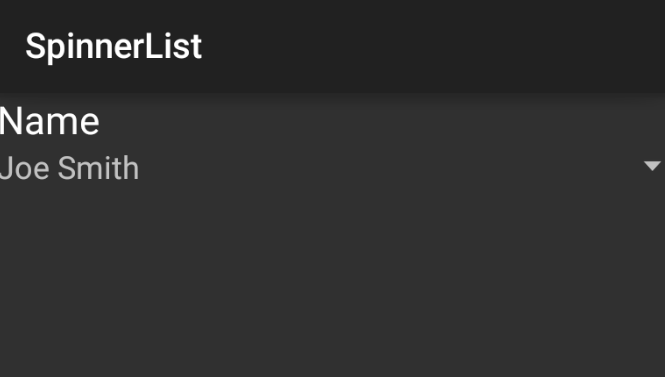
Here is how to create and populate a data adapter for a spinner.

string[] options = { "Joe Smith", "Carol Bennet", "Jake Bowman", "Bill Berritt", "Jen Mobley" };

ArrayAdapter adapter = new ArrayAdapter(this, Resource.Layout.text\_view\_for\_spinner, options);

spinner.Adapter = adapter;

A spinner automatically displays a result field for us, a name in this case.

Tapping the name field pops up the spinner.

The Spinner’s ItemSelected event fires when a selection is made.

spinner.ItemSelected += new EventHandler<AdapterView.ItemSelectedEventArgs>(spinner\_ItemSelected);

Create this method, which displays the selected name in a toast:

private void spinner\_ItemSelected(object sender, AdapterView.ItemSelectedEventArgs e)

This method retrieves the selected value and pops up a toast containing it.

var toastText = string.Format("Selection: {0}", spinner.GetItemAtPosition(e.Position));

Toast.MakeText(this, toastText, ToastLength.Long).Show();



The toast pops up with the selected name.

The Spinner automatically populates the original selection field with the selected value.

## Exercise 3: Create a ListView

Create a ListView that displays three items from a data model and add item selection to one or both projects. Note: in order to view these changes, you must remove the SpinnerList files from your solution.

### **ListView**

ListView is an Android collection of Views optimized for a large number of scrollable items. It contains several built-in views containing one or two lines of text and an image. Custom views can be constructed using LinearLayout at the cost of performance. Used in its basic form, ListView is simple and fast.

Create a list from a data model in three steps:

1. Data Model - Create a data model containing the list items.
2. Adapter - Create a custom list adapter to determine which fields to display in the list.
3. Activity - Populate the custom class and pass it into the custom list adapter constructor.
4. Create a data model class containing list items in a separate file called ListItem.cs.

public class ListItem

{

public string Title { get; set; }

public string Description { get; set; }

}

1. Create a new class called ListItemAdapter.cs. The custom list Adapter is built by inheriting from BaseAdapter and overriding Count, GetItemId, GetItem, and GetView. Create a private copy of the custom control, such as ListItem. Create a constructor which receives the Activity and populated data model as a parameter.

public class ListItemAdapter : BaseAdapter

{

private List<ListItem> itemList;

private Activity context;

public ListItemAdapter(Activity context, List<ListItem> items) : base()

{

this.context = context;

this.itemList = items;

}

public override int Count

{

get { return itemList.Count; }

}

public override long GetItemId(int position)

{

return position;

}

public override Java.Lang.Object GetItem(int position)

{

throw new NotImplementedException();

}

public override View GetView(int position, View convertView, ViewGroup parent)

{

View view = convertView;

if (view == null)

view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, null);

view.FindViewById<TextView>(Android.Resource.Id.Text1).Text = itemList[position].Title;

return view;

}

}

In GetView, always use this passed-in View and Inflate method technique to construct the cells.

1. Create an activity called List.cs subclassed from ListActivity. The adapter can now be bound to the ListActivity. In the ListActivity's OnCreate Method, populate the custom class and pass it into the ListAdapter's constructor. Assign the resulting adapter to the ListActivity.ListAdapter property.

[Activity(Label = "List", MainLauncher = true)]

public class List : ListActivity

{

protected override void OnCreate(Bundle bundle)

{

base.OnCreate(bundle);

List<ListItem> listItems = new List<ListItem> {

new ListItem {Title = "First", Description="1st item"},

new ListItem {Title = "Second", Description="2nd item"},

new ListItem {Title = "Third", Description="3rd item"}

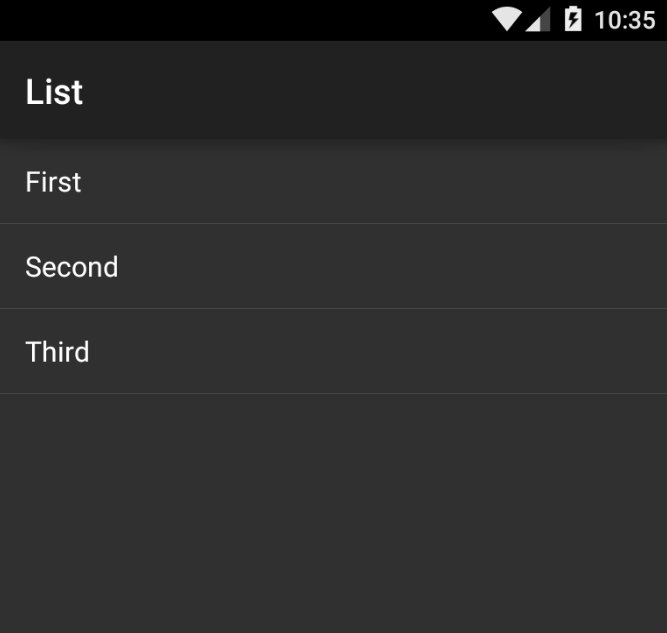
};

ListAdapter = new ListItemAdapter(this, listItems);

}

}

This will display a list of three items.



### **Cell Reuse**

When building a custom list adapter it is important to code for performance, which means cell reuse when possible. The GetView of the adapter constructs each row of the list as needed. The View parameter contains an existing row, if one exists. This should be used if possible. If it is null, then a new row must be created, using the Inflate method.

public override View GetView(int position, View convertView, ViewGroup parent)

{

View view = convertView;

if (view == null)

view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, null);

view.FindViewById<TextView>(Android.Resource.Id.Text1).Text = itemList[position].Title;

return view;

}

### **Item Selection**

User selection of a list item is handled by overriding the OnListItemClick method in the ListActivity. The selected item index is passed in through the parameter called position.

protected override void OnListItemClick(ListView l, View v, int position, long id)

{

var SelectedItem = listItems[position];

Android.Widget.Toast.MakeText(this, SelectedItem.Title, Android.Widget.ToastLength.Short).Show();

}

Remember to move the listItems declaration outside of the OnCreate method up into a class-level scope.

List<ListItem> listItems;

Touching a list item will now cause a toast to appear displaying the text of the item.

