

## Lab 01: ListView Control and Adapter Pattern

---

### Prerequisites

You will need a development environment, either a Mac or Windows PC with the Android SDK and Xamarin tools installed. We will be using the Android emulator to test the code we are building, so make sure to have a virtual device already configured and ready to run. See the [Xamarin.Android setup documentation](#) if you need help getting your environment setup.

### Downloads

Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is **Lab 01 Resources**. Make sure you have this folder before you begin.

### Lab Goals

The goal of this lab will be to examine the Android **ListView** functionality and learn how to work with it to display tables of information. We'll cover several related elements as part of our exercise:

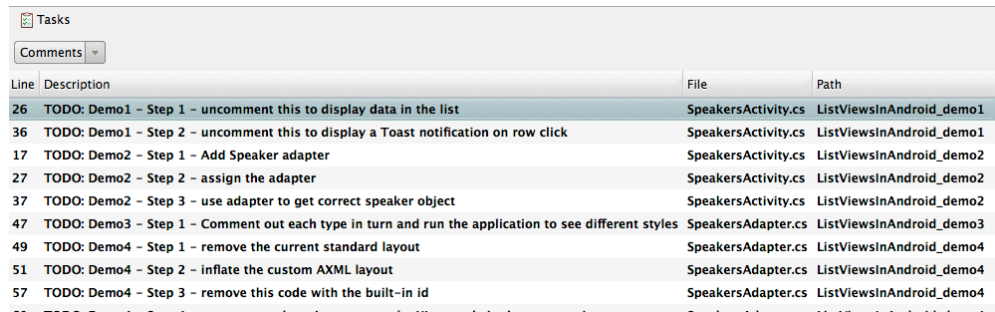
- **ListView** – A UI element that displays a scrollable table.
- **ListActivity** – A pre-built Android Activity, which includes a **ListView** as its primary UI display.
- **Adapter** – A non-visual class that generates the UI for each row in the associated **ListView** using an assigned data source. Android includes built-in adapter classes for simple scenarios, or you can build your own for custom data management.

The code you will work with has been structured to add functionality incrementally as you progress through the lab. There are six sample projects included in the solution to show off the basic features of the **ListView**:

- **ListViewsInAndroid\_demo1** – Implement a basic table using `ArrayAdapter<T>`.
- **ListViewsInAndroid\_demo2** – Use a custom `Adapter`
- **ListViewsInAndroid\_demo3** – Use different built-in row cell styles.
- **ListViewsInAndroid\_demo4** – Using a custom row cell style.
- **ListViewsInAndroid\_demo5** – Add fast scrolling and section indexes.
- **ListViewsInAndroid\_demo6** – Use ListViews for navigation.

The lab has been provided as a starter solution with most of the code already filled in for you – as you following along with the instructor you will make small changes for each step, either writing a little code or uncommenting a block of code. Most of these steps are clearly marked in the supplied solution with `// TODO:`

comments. These comments are picked up by Xamarin Studio and shown in the Task Pad, which you can make visible either by clicking the Tasks button in the status bar of the application, or through the **View > Pads > Tasks** menu item. When the Tasks Pad is open, it will look like this:



| Line | Description  | File                | Path                     |
|------|--|---------------------|--------------------------|
| 26   | TODO: Demo1 – Step 1 – uncomment this to display data in the list                                    | SpeakersActivity.cs | ListViewsInAndroid_demo1 |
| 36   | TODO: Demo1 – Step 2 – uncomment this to display a Toast notification on row click                   | SpeakersActivity.cs | ListViewsInAndroid_demo1 |
| 17   | TODO: Demo2 – Step 1 – Add Speaker adapter   | SpeakersActivity.cs | ListViewsInAndroid_demo2 |
| 27   | TODO: Demo2 – Step 2 – assign the adapter  | SpeakersActivity.cs | ListViewsInAndroid_demo2 |
| 37   | TODO: Demo2 – Step 3 – use adapter to get correct speaker object                                     | SpeakersActivity.cs | ListViewsInAndroid_demo2 |
| 47   | TODO: Demo3 – Step 1 – Comment out each type in turn and run the application to see different styles | SpeakersAdapter.cs  | ListViewsInAndroid_demo3 |
| 49   | TODO: Demo4 – Step 1 – remove the current standard layout  | SpeakersAdapter.cs  | ListViewsInAndroid_demo4 |
| 51   | TODO: Demo4 – Step 2 – inflate the custom AXML layout  | SpeakersAdapter.cs  | ListViewsInAndroid_demo4 |
| 57   | TODO: Demo4 – Step 3 – remove this code with the built-in id   | SpeakersAdapter.cs  | ListViewsInAndroid_demo4 |

You can quickly jump to the code by clicking on the task itself to keep up with the lecture as the instructor runs through this lab. If you need additional time to complete a task or need some help please let the instructor know – the goal is for you to work through this lab in the class itself.

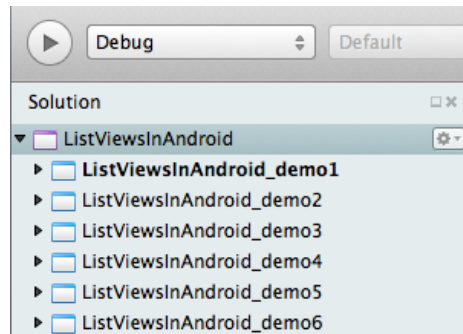
## Steps

### Open the Starting Solution

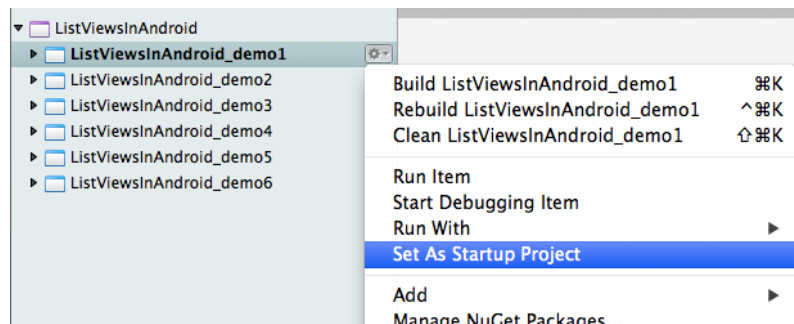
1. Launch **Xamarin Studio**.
2. Click **Open...** on the Xamarin Studio Welcome screen and navigate to the Lab 01 Resources folder included with this document.
3. Locate the ListViewsAndroid\_Begin folder, inside this folder you will find a `ListViewsInAndroid.sln` file – double click on this file to open the starter solution.
4. This solution has 6 demo projects we'll be using to learn about ListViews in Android. All of the code is already present in the solution but some of the key parts have been commented out so we can try different options during the lab. We'll take the approach of doing the lab steps first so we can see the result in the Android Emulator and then we'll go over the code together

### Populating a ListView with Data

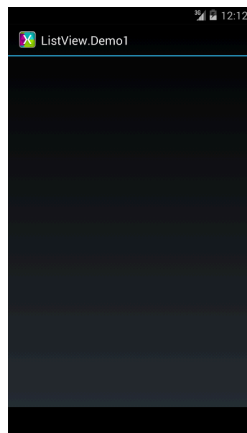
1. The project **ListViewsInAndroid\_demo1** should be setup as the *Startup Project* and the project name should be in bold type as shown below:



If the **ListViewsInAndroid\_demo1** project does not appear in bold type select **ListViewsInAndroid\_demo1** in the **Solution Pad** and select **Set As Startup Project** from the Action Button that appears to the right of the entry when selected as shown below:



5. Expand the ListViewsInAndroid\_demo1 project.
6. Click the Play button in the toolbar to build and run the project in the emulator. Once the program launches it should display an empty screen:



7. Start by using the built-in ArrayAdapter class to bind a collection of data to the ListView control. Open the `SpeakersActivity.cs` file and find the comment `TODO: Demo1 - Step 1 -` uncomment this to display data in the list and uncomment the two lines that follow it.

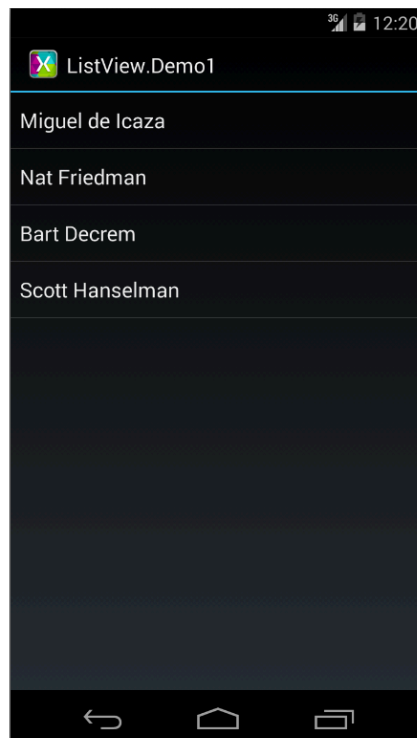
An easy way to comment and uncomment code in Xamarin Studio is to use the **Command + /** hotkey, or right click and select “Toggle Line Comment(s)”.

When you are finished the code should look something like:

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    //TODO: Demo1 - Step 1 -
    uncomment this to display data in the list
    items = new string[] { "Miguel de Icaza", "Nat Friedman",
                          "Bart Decrem", "Scott Hanselman" };
    ListAdapter = new ArrayAdapter<String>(this,
    Android.Resource.Layout.SimpleListItem1, items);
}
```

8. Build and run the application again to see the results of assigning an adapter:



Tapping on the items should briefly highlight the row, but there will be no action taken by the program in response. In addition, while scrolling is built into the

ListView, since we don't have enough items to fill our screen you won't be able to scroll the data.

## Responding to User Interactions

1. To respond to touch events on an item in the list, we need to implement the `OnListItemClick` method in the `SpeakersActivity`. This is an override available when you use the `ListActivity` type, but there are also events exposed on the `ListView` type if you use one directly.

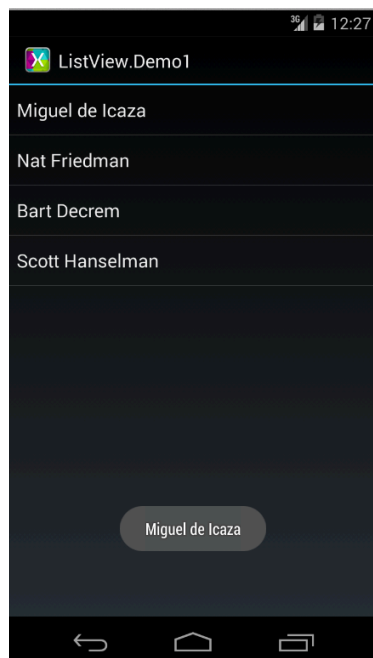
There is already an implementation present in our source file (`SpeakersActivity.cs`) – locate the comment: `TODO: Demo1 - Step 2 - uncomment this to display a Toast notification on row click`

Uncomment the two lines below this to show a toast popup message when the click occurs. Your code should look something like:

```
protected override void OnListItemClick(ListView l, View v,
                                         int position, long id)
{
    //TODO: Demo1 - Step 2 -
    uncomment this to display a Toast notification on row click
    var t = items[position];
    Android.Widget.Toast.MakeText(this, t, Android.Widget.ToastLength.Short).Show();
}
```

**Note:** This is just an example behavior – you could also navigate to a new screen as you did in the introduction module or provide some other programmed behavior when you tap on an item.

2. Run the application again and tap on a row to display the toast notification:



## Creating a Custom Adapter

1. Select **ListViewsInAndroid\_demo2** in the **Solution Pad** and choose “Set As Startup Project” from the Action Button, once you have completed this, the project should be in bold indicating it is the startup project.
2. Expand the **ListViewsInAndroid\_demo2** project to display all the source files.

In this project, we will be using a list of **Speaker** objects to display our data, instead of a set of strings. This makes it slightly more complicated, as the built-in **Adapter<T>** does not know how to display **Speaker** objects visually. For that we will need to use a custom **Adapter**.

3. To create a custom Adapter, you will typically derive from an existing adapter class – in this case we’ll use the **BaseAdapter<T>** which provides no expectation for our data.

An implementation has been included in the project already; named **SpeakerAdapter** – go ahead and open the corresponding source file (**SpeakerAdapter.cs**) to look through the code. It has 4 overloaded methods that it implements for the **BaseAdapter<T>** base class:

- a. **GetItemId** – returns a unique row identifier for a given index – in this case we are returning the row index itself.
  - b. **Count** – returns the total number of elements for the ListView to display.
  - c. **GetView** – creates a single row’s View to display.
  - d. **Indexer** – provides easy access to the underlying **Speaker** object for a given row index.
4. Now, let’s use the new adapter. Open the **SpeakersActivity.cs** file and find the comment

```
// TODO: Demo2 - Step 1 - Speaker adapter
```

And uncomment the adapter field below it.

5. Next, in the **OnCreate** method, delete the current implementation which assigns the **ListAdapter** property and uncomment the code under the comment:

```
// TODO: Demo2 - Step 2: assign the adapter
```

6. Finally, uncomment the implementation in **OnListItemClick** – this will reach into the adapter to retrieve the current speaker object and then display the name as a popup. There is a **//TODO** comment here as well (Step 3).

Your completed code should look something like:

```
public class SpeakersActivity : ListActivity
{
    //TODO: Demo2 - Step 1 - Add Speaker adapter
    SpeakersAdapter adapter;
```

```

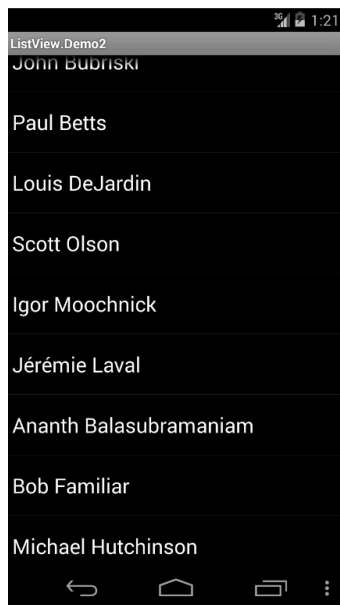
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    //TODO: Demo2 - Step 2 - assign the adapter
    adapter = new SpeakersAdapter(this, Speakers.GetSpeakerData())
;
    ListAdapter = adapter;
}

/// <summary>
/// Demonstrates how to handle a row click
/// </summary>
protected override void OnListItemClick(ListView l, View v, int po
sition,
                                     long id)
{
    //TODO: Demo2 - Step 3 -
    use adapter to get correct speaker object
    var t = adapter[position].Name;
    Android.Widget.Toast.MakeText(this, t, Android.Widget.ToastLen
gth
                                     .Short).Show();
}
}

```

7. Run the application to see our custom adapter – it should now have a lot more data and you should be able to scroll the items as shown below:

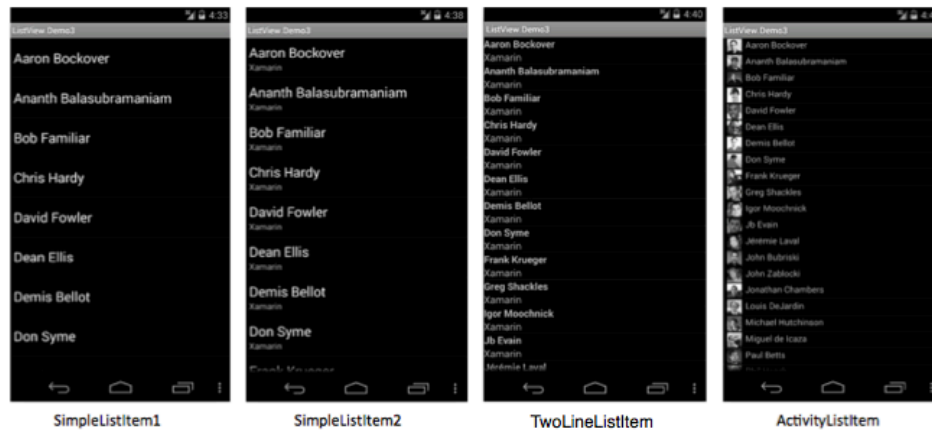


## Changing the Cell Style

1. Switch to the third demo project in the solution. You should now have **ListViewsInAndroid\_demo3** selected as the active project. (Remember: the project name is bolded when it is the active project so it's easy to spot).

- Open the **SpeakersAdapter.cs** file – this is where all of your code changes will occur in this project. The `GetView` method is used to build the view to display a single row of data.

Android has several built-in layout styles for rows to select from. Here are the layouts you will use:



- Locate the comment in the `GetView` method:

```
// TODO: Demo3 - Part1 -
Comment out each type in turn and run the application to see different styles
```

- Just build and run the application to see the `SimpleListItem1` style.
- Comment out the view creation line and uncomment the next one in turn and run it to see the `SimpleListItem2` style.

```
if (view == null) {
    //TODO: Demo3 Part 1: Comment out each type in turn and run the application to see different styles
    view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem1, null);
    // view = context.LayoutInflater.Inflate(Android.Resource.Layout.SimpleListItem2, null);
    // view = context.LayoutInflater.Inflate(Android.Resource.Layout.TwoLineListItem, null);
    // view = context.LayoutInflater.Inflate(Android.Resource.Layout.ActivityListItem, null);
}
```

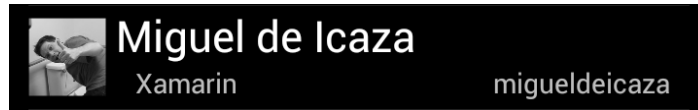
- Do the same for each of the styles to see what each one looks like. Notice that depending on the style you select you have different UI controls you can fill in – the code just below the view creation checks for different text and image elements and populates them if present.

Many applications can display exactly what they want using the built-in styles, but if you need a custom visual display for the row, you will need to create your view in code or through an XML file.



## Creating a Custom Cell View

1. Switch to the next project in the solution – you should now have **ListViewsInAndroid\_demo4** set as the active project.
2. In this project, we have a custom `speaker_row.xml` file in the `Resources/Layout` folder – go ahead and open it up to look at the contents. It includes an image for a headshot, the speaker's name, company and Twitter handle. Open the AXML file in the resources and identify each visual element in the layout – the UI is shown below:



You should be able to determine that the image is displayed with an **ImageView** control, the user's name with a **Text (Large)** control and the company name and Twitter handle are both displayed using a **Text (Small)** control.

3. Next, open the **SpeakersAdapter.cs** file – as before, this is where all our changes will occur. Locate the **GetView** method that generates the row visuals. There are four steps present here to activate our custom visual – each marked with a `TODO` comment (Step 1 – 4). Go ahead and do each step.

```
//TODO: Demo4 - Step 1 - remove the current standard layout
//TODO: Demo4 - Step 2 - inflate the custom AXML layout
//TODO: Demo4 - Step 3 - remove this code with the built-in id
//TODO: Demo4 - Step 4 - uncomment these lines to set the UI controls
in the custom view
```

4. Build and run the application to see the custom visuals:



## Fast Scrolling

1. Switch to the next project in the solution – you should now have **ListViewsInAndroid\_demo5** selected as the active project.
2. Build and run the application – this version has quite a few more speakers in it (duplicated entries to add more data).



Try scrolling through the list to see how many there are. When you have a lot of data in the ListView scrolling becomes tedious – but there are a couple of tricks you can do. The first is to turn on *Fast Scrolling*. This will add a thumb to the scroll experience so you can quickly slide through the list. This is activated by setting the **FastScrollEnabled** property.

3. Open up the **SpeakersActivity.cs** file and locate the comment

```
//TODO: Demo5 - Step 1 - uncomment to enable fast scrolling
```

Uncomment the line below it to activate fast scrolling. The code should look like:

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    adapter = new SpeakersAdapter(this, Speakers.GetSpeakerData());
    ListView.Adapter = adapter;

    //TODO: Demo5 - Step 1 - uncomment to enable fast scrolling
    ListView.FastScrollEnabled = true;
}
```

4. Run the app again and try scrolling through the data – now you should see a thumb on the right hand side which you can grab and quickly jump to different sections of the list:



## Adding a Fast Scrolling index visual

Fast Scrolling is great because it lets you quickly jump through entries – but it can be hard to tell exactly where you are in the list. To help with that, you can turn on section indexes by having the Adapter implement the `ISectionIndexer` interface. When the `ListView` sees this interface implementation on the Adapter, it will use it to provide sections and visual indexes for the items as you scroll.

5. The code for `ISectionIndexer` has already been provided for you in the Adapter implementation but it has been commented out. Open the **SpeakersAdapter.cs** file and locate the comment labeled:

```
//TODO: Demo5 - Step 2 - Implement ISectionIndexer
```

6. It starts a section of the source file that has been completely commented out that contains several fields and methods. Go ahead and remove the start and close comments around the entire block. This will uncomment several methods:
  - a. **SetupIndex** – this is an initializer method we will call from the constructor to create the indexes. The algorithm used here is very simple – it just uses the first character of the speaker's name to provide an index, depending on your data you might use some other implementation.
  - b. **GetPositionForSection** – this is a lookup method that tells the **ListView** the row where the specified section begins.
  - c. **GetSectionForPosition** – this is a lookup method that tells the **ListView** which section a specified row belongs to.
  - d. **GetSections** – this returns an array of objects representing the sections for the list – Android will call `ToString` to create a string for each item. It must be `Java.Lang.Object` array and in this implementation the code returns a collection of strings.

- Next, indicate that the adapter implements the interface by adding it to the class definition, place it directly after the `BaseAdapter<Speaker>` base class definition, the code should look something like:

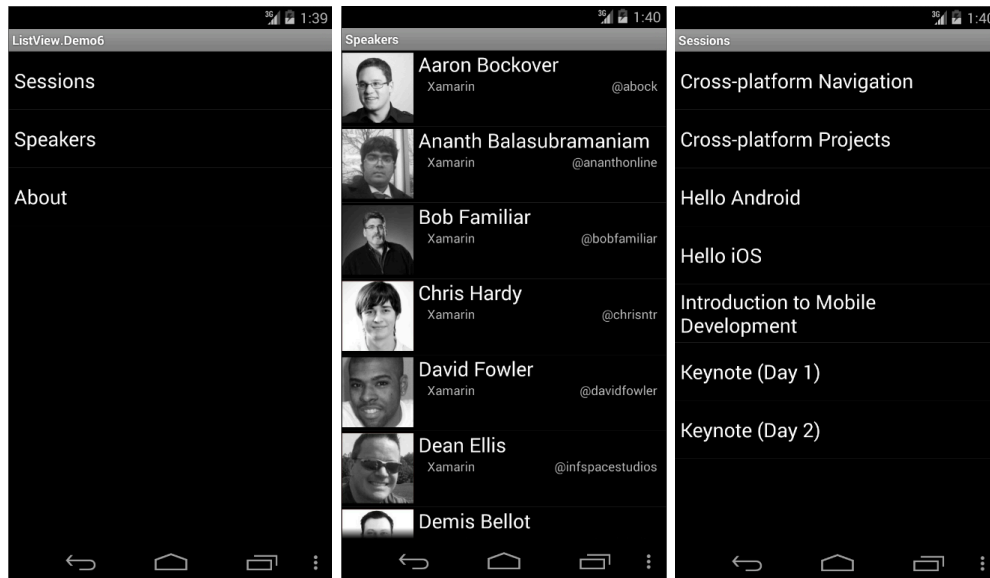
```
public class SpeakersAdapter: BaseAdapter<Speaker>, ISectionIndexer
```

- Finally, uncomment the call to **SetupIndex** in the constructor, it is marked with the `//TODO: Demo5 - Step 3 - setup the indexer support` comment.
- Build and run the application one last time and scroll using the thumb. You should see the section indexer visual display as you scroll:



## Using ListViews for Navigation

- Switch to the final project, which is **ListViewsInAndroid\_demo6**.
- Build and run the application – it should present an initial **ListView** with three choices – selecting one of these choices navigates to a new screen and you can then go back using either the **Back** button on the screen (for newer Android devices or the KitKat emulator), or through a hardware **Back** button:



3. Open the **MenuActivity.cs** – this is the starting activity and displays the initial menu. In the `OnCreate` override you will find code to setup the main menu, and in the `OnListItemClick` you will find the code to navigate to the next screen as shown below:

```
protected override void OnListItemClick(ListView l, View v, int position,
                                         long id)
{
    //TODO: Demo6 - process the selection
    Intent nextScreen = null;
    switch (position)
    {
        case 0:
            nextScreen = new Intent(this, typeof(SessionsActivity));
            break;
        case 1:
            nextScreen = new Intent(this, typeof(SpeakersActivity));
            break;
        case 2:
            nextScreen = new Intent(this, typeof(AboutActivity));
            break;
    }

    if (nextScreen != null) {
        StartActivity(nextScreen);
    }
}
```

4. The rest of the screens are done in exactly the same fashion and are slight variations of what you've done in this lab but take a moment and go through the screens to see the flow.

## Summary

We covered a lot of ground in this tutorial. The `ListView` class provides a flexible way to present data, whether it is a short menu or a long, scrolling list. The first thing we explored was how to use `ListActivity` with the built-in Adapter `ArrayAdapter<T>`. Then, by subclassing `BaseAdapter`, we learned how to customize the appearance of the `ListView` to display a more complex data type.

Finally, we turned on fast scrolling and section indexes, and learned how to use **ListViews** to create a basic navigation hierarchy.