Hello, Android Multiscreen

Handing Navigation with Xamarin.Android

In this two-part guide, we expand the Phoneword application we created in the Hello, Android guide to handle a second screen. Along the way we'll introduce the basic Android Application Building Blocks and dive deeper into Android architecture as we develop a better understanding of Android application structure and functionality.

Hello, Android Multiscreen Deep Dive

In the Hello, Android Multiscreen Quickstart, we built and ran our first multi-screen Xamarin.Android application. Now it's time to develop a deeper understanding of Android navigation and architecture so that we can build more complex applications.

In this guide we explore more advanced Android architecture as we introduce Android *Application Building Blocks*, dive into Android navigation with *Intents*, and explore Android hardware navigation options. We will dissect the new additions to our Phoneword app as we develop a more holistic view of our application's relationship with the operating system and with other applications.

Android Architecture Basics

In the Hello, Android Deep Dive, we learned that Android applications are unique programs because they lack a single entry point. Instead, the operating system (or another application) starts any one of the application's registered Activies, which in turn starts the process for the application.

In this deep dive into Android architecture, we're going to expand our understanding of how Android applications are constructed by introducing the Android *Application Building Blocks* and their functions.

Android Application Blocks

An Android application consists of a collection of special Android classes called *Application Blocks* bundled together with any number of app resources - images, themes, helper classes, etc. - and coordinated by an XML file called the *Android Manifest*.

Application Blocks form the backbone of Android applications because they allow us to do things we couldn't normally accomplish with a regular class. The two most important ones are Activities and Services:

- **Activity** - An Activity corresponds to a screen with a user interface, and it is conceptually similar to a web page in a web application. For example, in a newsfeed application, the login screen would be the first Activity, the scrollable list of news items would be another Activity, and the details page for each item would be a third. You can learn more about Activities in the Activity Lifecycle guide.
- **Service** - Android Services support Activities by taking over long-running tasks and running them in the background. Services don't have a user interface and are used to handle tasks that aren't tied to screens - for example, playing a song in the background or uploading photos to a server. For more information on Services, check out the Creating Services and Android Services guides.

An Android application may not use all types of Blocks, and often has several Blocks of one type. For example, our Phoneword application from the Hello, Android Quickstart was composed of just one Activity (screen) and resource files. A simple music player app might have several Activities and a Service for playing music when the app is in the background.

Intents

Another fundamental concept in Android applications is the *Intent*. Android is designed around the *principle of least privilege* - applications have access to only the Blocks they require to work, and limited access to the Blocks that make up the operating system or other applications. Similarly, Blocks are *loosely-coupled* - designed to have little knowledge of and limited access to other Blocks, even blocks that are part of the same application.

In order to communicate, Application Blocks send asynchronous messages called *Intents* back and forth. Intents contain information about the receiving Block and sometimes some data. An Intent sent from one App Component triggers something to happen in another App Component, binding the two App Components and allowing them to communicate. By sending Intents back and forth, we can get Blocks to coordinate complex actions such as launching the camera app to take and save, gathering location information, or navigating from one screen to the next.

AndroidManifest.XML

When we add a Block to our application, it is registered with a special XML file called the *Android Manifest*. The Manifest keeps track of all Application Blocks in an application, as well as version requirements, permissions, and linked libraries - everything the operating system needs to know for our application to run. The *Android Manifest* also works with Activities and Intents to control what actions are appropriate for a given Activity. These advanced features of the Android Manifest are covered in the Working with the Android Manifest guide.

In the single-screen version of the Phoneword application, we only had one Activity, one Intent, and the `AndroidManifest.xml`, alongside additional resources like icons. In the multi-screen version of Phoneword, we added an additional Activity, which we launched from the first Activity using an Intent. In the next section, we'll explore how Intents help us create navigation in Android applications.

Android Navigation

We used Intents twice in our Phoneword application – once to navigate between screens, and once to place a phone call. Let's dive into this code to see how Intents work and understand their role in Android navigation.

Launching a Second Activity with an Intent

In the Phoneword application, we use an Intent to launch a second screen (Activity). We start by creating an Intent and we pass in the current *Context* (`this`, referring to the current *Context*) and the type of Application Block we're looking for (`CallHistoryActivity`):

```
Intent intent = new Intent(this, typeof(CallHistoryActivity));
```

The *Context* is an interface to global information about the application environment - it lets newly created objects know what's going on with the application. If we think of an Intent as a message, we are providing the name of the message recipient (`CallHistoryActivity`) and the receiver's address (`Context`).

Android gives us an option to attach simple data to an Intent (complex data is handled differently). In the Phoneword example, we use `PutStringArrayExtra` to attach a list of phone numbers to our Intent, and then call `StartActivity` on the recipient of the Intent. The completed code looks like this:

```
callHistoryButton.Click += (sender, e) =>

{

    var intent = new Intent(this, typeof(CallHistoryActivity));

    intent.PutStringArrayListExtra("phone_numbers", _phoneNumbers);

    StartActivity(intent);

};
```

Launching Another Application with an Intent

In the example above we used an Intent to launch a second Activity in the same application, but Android makes it just as easy to launch an Activity from *another* application. In fact, we've already done it in our Phoneword application. When the user pressed the Call Button, we used the following code to dial the phone number:

```
Intent callIntent = new Intent(Intent.ActionCall);

callIntent.SetData(Android.Net.Uri.Parse("tel:" + translatedNumber));

StartActivity(callIntent);
```

When we create the Intent, we pass in `Intent.ActionCall`. `ActionCall` is a special Intent that launches the Android phone application and passes in a phone number to dial. Then we use `SetData` to attach the phone number to the Intent. When we call `StartActivity` and pass in the `callIntent`, we bring up the call screen of the system phone application. This loads the system phone application's process into memory, therefore launching the system phone app.

Additional Concepts Introduced in Phoneword

The Phoneword application introduced several concepts not covered in this guide. These concepts include:

- **String Resources** – In our Phoneword application, we set the text of the `CallHistoryButton` to "`@string/callHistory`". The @`string` syntax means that the string's value is stored in the *String Resources File*, *Strings.xml*. We entered the following value for the `callHistory` string in *Strings.xml*:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

   <string name="callHistory">Call History</string>

</resources>
```

  For more information on string resources and other Android resources, refer to the Android Resources guide.

- **ListView and ArrayAdapter** – A *ListView* is a UI component that provides a simple way to present a scrolling list of rows. A `ListView` instance requires an *Adapter* to feed it with data contained in row views. We used the following line of code to populate the user interface of our `CallHistoryController`:

```
this.ListAdapter       =       new       ArrayAdapter<string>(this,
Android.Resource.Layout.SimpleListItem1, phoneNumbers);
```

ListViews and Adapters are beyond the scope of this document, but they are covered in the very comprehensive ListViews and Adapters guide. Part 2 of ListViews and Adapters guide deals specifically with using built-in `ListActivity` and `ArrayAdapter` classes to create and populate a `ListView` without defining a custom layout, like we did in the Phoneword example.

Summary

Congratulations, you've completed your first multi-screen Android application!

In this guide we introduced *Android Application Building Blocks* and *Intents*, and used them to build a multi-screened Android application. You now have the solid foundation you need to start developing your own Xamarin.Android applications.

Next, let's learn to build cross-platform applications with Xamarin with the Building Cross-Platform Applications guides.