Lab 02: Creating multi-screen Android apps

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:

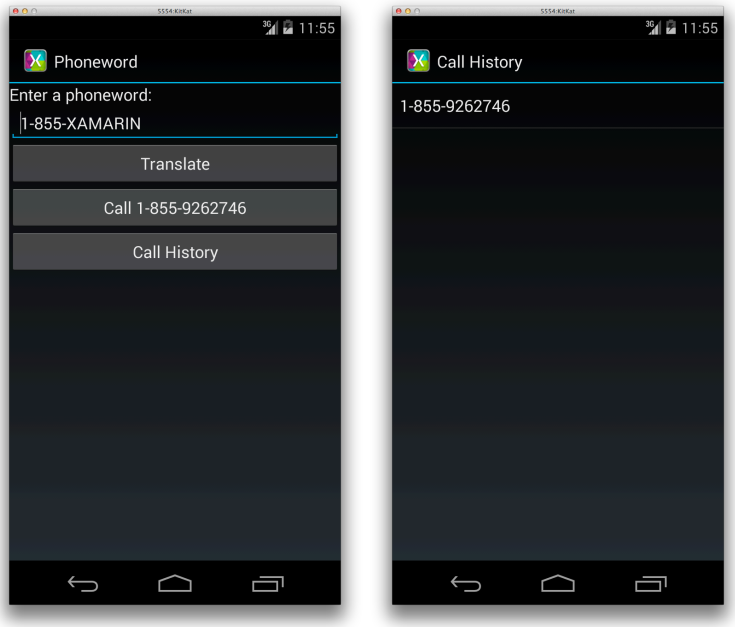
http://docs.xamarin.com/guides/android/getting_started/installation/

Downloads

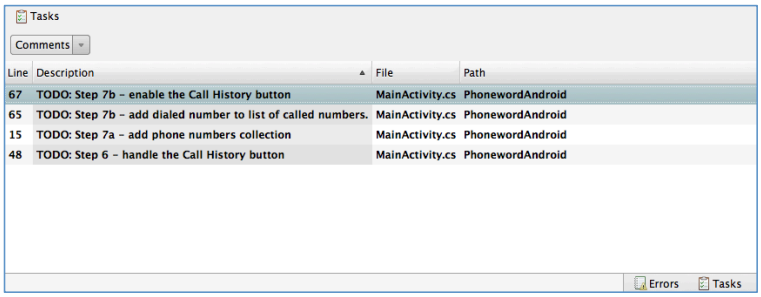
Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is **Lab 02 Resources**. Make sure you have this folder before you begin.

Lab Goals

In this lab we'll explore how to create multi-screened Android applications using Activities and Intents. We'll do this by adding support for navigation between multiple screens by extending the **PhonewordAndroid** application we created in Lab 1 to include a second screen that contains the call history, as illustrated below:



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:



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.

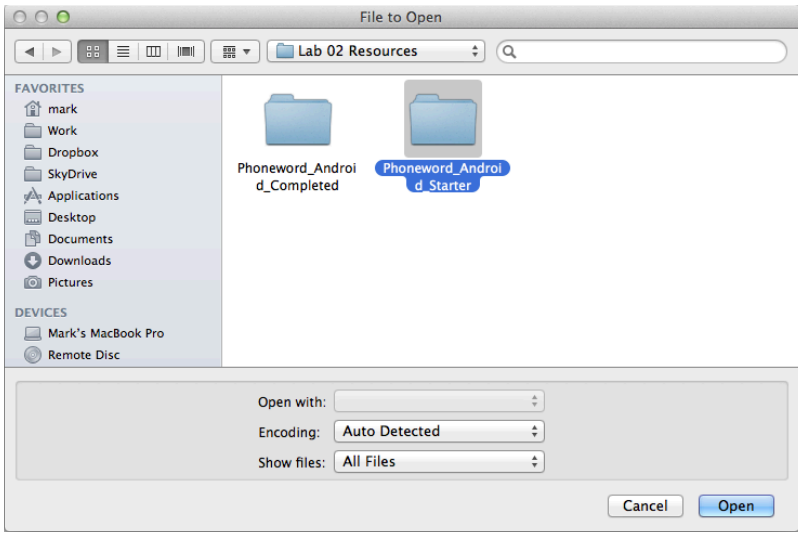
Steps

Open the Starter Solution

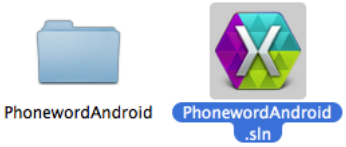
1. Launch **Xamarin Studio**.
2. Click **Open...** on the Xamarin Studio Welcome screen and navigate to the **Lab 02 Resources** folder included with this document.

Craig Dunn 1/13/14 3:44 PM
Comment [1]: I think we use 'directory'? we should standardize anyway

3. Locate the **Phoneword_Android_Starter** folder – make sure it's the starter and not the completed folder:



4. Inside the Phoneword_Android_Starter folder you will find a `PhonewordAndroid.sln` file – double click on this file to open the starter solution:

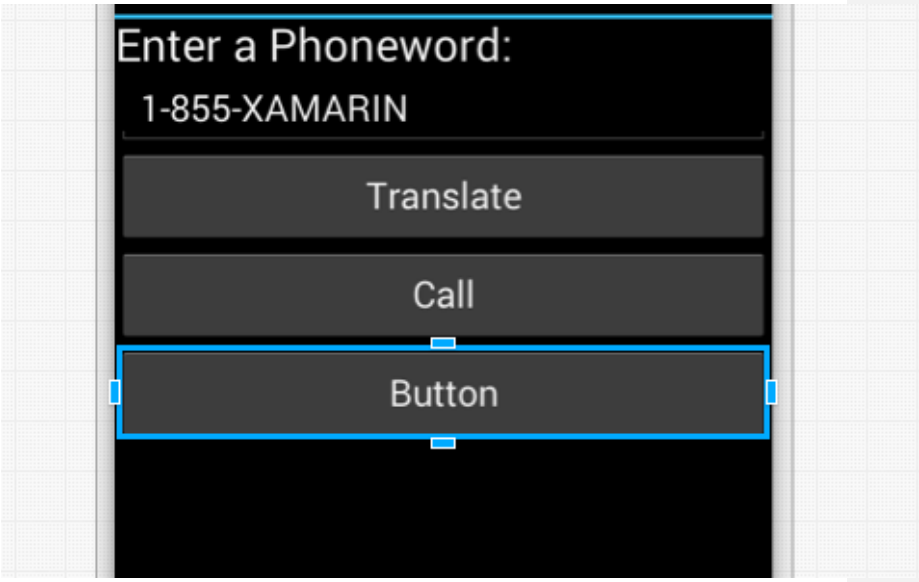


5. Go ahead and build and run the application in the emulator to make sure it compiles and your environment is ready. Let the instructor know if you have any trouble.

Adding the Call History Button

We are going to add a **Call History** button to the main screen of our **PhonewordAndroid** app to display any calls the user has initiated from within our app.

1. Open the **Main.axml** layout file in the **Resources/layout** folder, just expand the folders and then double-click on the file to open it in the designer.
2. Drag a new button onto the design surface directly under the existing **Call** button, it should look something like:



3. Make the following changes to the new button on the **Property Pad > Widget Tab**:

Property	Value
Id	@+id/CallHistoryButton
Text	@string/callHistory

4. Notice that the Button displays exactly what you placed into the `Text` property – we want this to come from our string resources so open the **Strings.xml** file in the **Resources/values** folder.
5. Add a new string into the file with the name `callHistory` and the value **Call History**, it should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Click Me!</string>
```

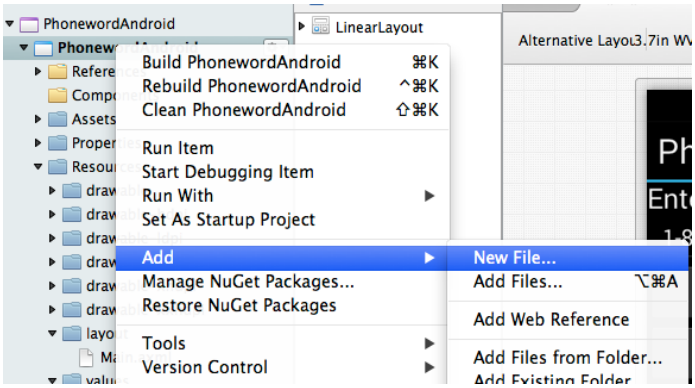
```
    <string name="app_name">PhonewordAndroid</string>
    <string name="callHistory">Call History</string>
</resources>
```

6. Switch back to the **Main.axml** file and verify that the button now displays the text **Call History**.
7. On the **Property Pad** > **Behavior** tab, change the enabled state of the **Button** to be **false**. The **Button** in the designer should appear disabled.
8. **File > Save All** (or **Shift+Command+S**).

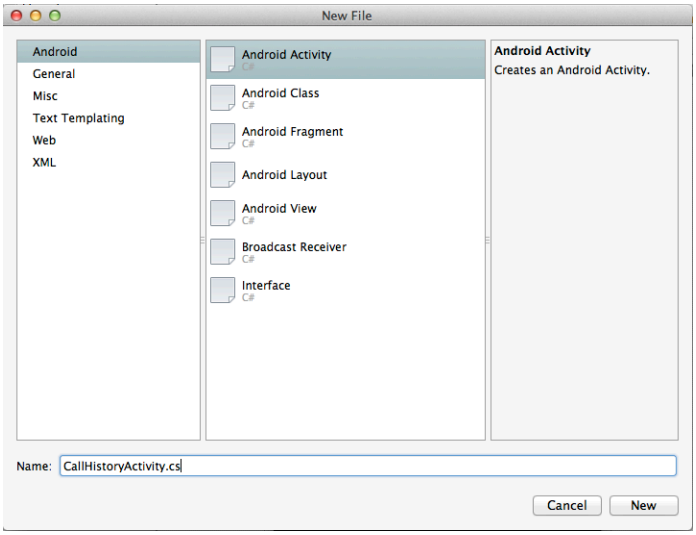
Adding the Call History Activity

We will display a list of call placed within our application through a `ListActivity` titled **Call History** which will be shown when the user clicks on the **Call History Button** we just added. The `ListActivity` will show call activity as shown in the screen shot below:

1. Right-click on the **PhonewordAndroid** project and select **Add > New File**:



2. From the New File dialog, select the **Android Activity** item and set the **Name** to **CallHistoryActivity.cs**. Click **New** to create the activity.



3. Open the **CallHistoryActivity.cs** file, it currently derives from the base `Activity` type, so change this to be `ListActivity` that displays lists of items.

```
[Activity(Label = "CallHistoryActivity")]
public class CallHistoryActivity : ListActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);

        // Create your application here
    }
}
```

4. Next, change the `Label` property on the `[Activity]` attribute to be **@string/callHistory** so we get the proper title bar caption.

```
[Activity(Label = "@string/callHistory")]
public class CallHistoryActivity : ListActivity
{
```

5. Next, modify the `OnCreate` method with the following code to get a string array passed from the caller and display it as the data for the `ListActivity`.

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

    var phoneNumbers = Intent.Extras.GetStringArrayList("phone_numbers")
                       ?? new string[0];
    this.ListAdapter = new ArrayAdapter<string>(this,
                       Android.Resource.Layout.SimpleListItem1, phoneNumbers);
}
```

6. Open the **MainActivity.cs** source file and locate the `OnCreate` method – at the bottom you will find a comment: `// TODO: Step 1 – handle the Call History button`. Go ahead and uncomment the code below it to handle the **CallHistoryButton** `Click` event and display the **CallHistoryActivity**. The code should look something like:

```
// TODO: Step 1 - handle the Call History button
Button CallHistoryButton = FindViewById<Button>(Resource.Id.CallHistoryButton);
CallHistoryButton.Click += (sender, e) =>
{
   var intent = new Intent(this, typeof(CallHistoryActivity));
   intent.PutStringArrayListExtra("phone_numbers", _phoneNumbers);
   StartActivity(intent);
};
```

7. Next, we need a collection to hold the call history, and some code to add each dialed number to that collection.
   a. Locate the comment `// TODO: Step 2 – add phone numbers collection`, just below this you will find a `List<string>` named `phoneNumbers` which is commented out – uncomment this.
   b. In the **callButton** `Click` handler, uncomment the code to add the translated number to the call history collection and to enable the **Call History Button**. There is a `TODO` comment (Step 3a and Step 3b) before both of these lines of code.

Testing our Application

1. Click the **Play** button to build and run our app.
2. Click the **Translate** button, followed by the **Call 1-855-9262746** button.
3. Click the **Call** button to dial the call and then end the simulated call.
4. Click the **Call History** button. We see that navigating to the call history after dialing shows the list of numbers dialed:

Craig Dunn 1/13/14 3:44 PM
Comment [2]: Parameter? This is a minor point – Property makes sense too – but a coder looking at the sample is going to recognize the named parameter as such, rather than an object property?

Mark Smith 1/13/14 3:49 PM
Comment [3]: It appears to be a property on the attribute class – not a named parameter for the constructor: http://androidapi.xamarin.com/?link=T%3aAndroid.App.ActivityAttribute%2fP

Craig Dunn 1/13/14 3:44 PM
Comment [4]: Why _ separated versus InitialCaps or camelCase? Seems like we have lots of different 'identifier standards' (apols if you inherited it from the sample – I'm just sayin'…)

Mark Smith 1/13/14 3:50 PM
Comment [5]: Yes, I inherited the code from the Evolve sample.. same for below.

Craig Dunn 1/13/14 3:44 PM
Comment [6]: ditto for leading underscore – not part of our coding guidelines

Summary

Congratulations! You have successfully built a multi-screen Android application by adding a new `Activity` to the project and using an `Intent` to activate that activity and pass it some data to process.