Extracted from:

Hello, Android

Introducing Google's Mobile Development Platform, 2nd Edition

The Pragmatic Programmers

Android 1.5 & 1.6

Hello, Android

Introducing Google's Mobile Development Platform

Second Edition

Ed Burnette

Edited by Susannah Davidson Pfalzer

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

http://www.pragprog.com

Printed in the United States of America.

ISBN-10: 1-934356-49-2

ISBN-13: 978-1-934356-49-4

Printed on acid-free paper.

P1.0 printing, October 2009

Version: 2009-10-6

Figure 3.1: The Sudoku example program for Android

Now let's see how we can use this information to create the Sudoku opening screen.

3.3 Creating the Opening Screen

We'll start with a skeleton Android program created by the Eclipse plug-in. Just as you did in Section 1.2, *Creating Your First Program*, on page 22, create a new "Hello, Android" project, but this time use the following values:

```
Project name: Sudoku
Build Target: Android 1.6
Application name: Sudoku
Package name: org.example.sudoku
Create Activity: Sudoku
```

In a real program, of course, you would use your own names here. The package name is particularly important. Each application in the system

must have a unique package name. Once you choose a package name, it's a little tricky to change it because it's used in so many places.

I like to keep the Android emulator window up all the time and run the program after every change, since it takes only a few seconds. If you do that and run the program now, you'll see a blank screen that just contains the words "Hello World, Sudoku." The first order of business is to change that into an opening screen for the game, with buttons to let the player start a new game, continue a previous one, get information about the game, or exit. So, what do we have to change to do that?

As discussed in Chapter 2, *Key Concepts*, on page 29, Android applications are a loose collection of activities, each of which define a user interface screen. When you create the Sudoku project, the Android plug-in makes a single activity for you in Sudoku.java:

Download Sudokuv0/src/org/example/sudoku/Sudoku.java

```
package org.example.sudoku;

import android.app.Activity;
import android.os.Bundle;

public class Sudoku extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Android calls the onCreate() method of your activity to initialize it. The call to setContentView() fills in the contents of the activity's screen with an Android view widget.

We could have used several lines of Java code, and possibly another class or two, to define the user interface procedurally. But instead, the plug-in chose the declarative route, and we'll continue along those lines. In the previous code, R.layout.main is a resource identifier that refers to the main.xml file in the res/layout directory (see Figure 3.2, on the following page). main.xml declares the user interface in XML, so that's the file we need to modify. At runtime, Android parses and instantiates (*inflates*) the resource defined there and sets it as the view for the current activity.

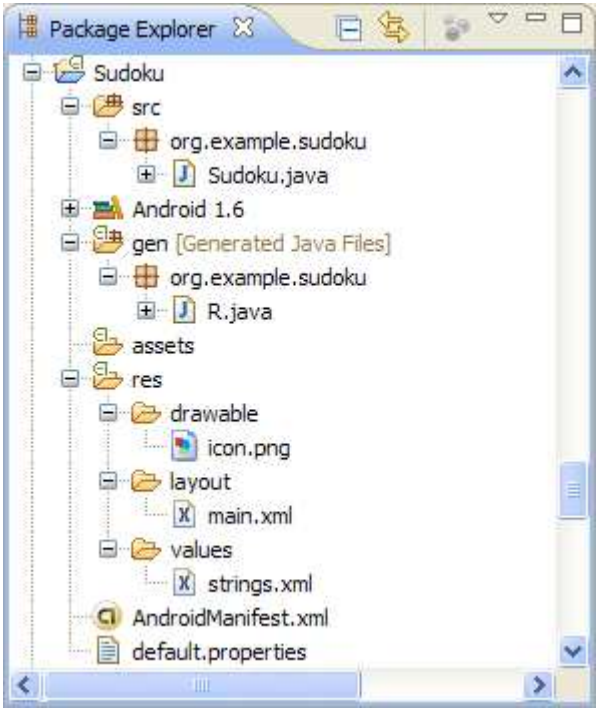

Figure 3.2: Initial resources in the Sudoku project

It's important to note that the R class is managed automatically by the Android Eclipse plug-in. When you put a file anywhere in the res directory, the plug-in notices the change and adds resource IDs in R.java in the gen directory for you. If you remove or change a resource file, R.java is kept in sync. If you bring up the file in the editor, it will look something like this:

Download Sudokuv0/gen/org/example/sudoku/R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package org.example.sudoku;
```

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

The hex numbers are just integers that the Android resource manager uses to load the real data, the strings, and the other assets that are compiled into your package. You don't need to worry about their values. Just keep in mind that they are handles that refer to the data, not the objects that contain the data. Those objects won't be inflated until they are needed. Note that almost every Android program, including the base Android framework itself, has an R class. See the online documentation on android.R for all the built-in resources you can use.[1]

So, now we know we have to modify main.xml. Let's dissect the original definition to see what we have to change. Double-click main.xml in Eclipse to open it. Depending on how you have Eclipse set up, you may see either a visual layout editor or an XML editor. In current versions of ADT, the visual layout editor isn't that useful, so click main.xml or Source tab at the bottom to see the XML. The first line of main.xml is as follows:

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

All Android XML files start with this line. It just tells the compiler that the file is XML format, in UTF-8 encoding. UTF-8 is almost exactly like regular ASCII text, except it has escape codes for non-ASCII characters such as Japanese glyphs.

Next we see a reference to *<LinearLayout>*:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <!-- ... -->
</LinearLayout>
```

1. http://d.android.com/reference/android/R.html

Joe Asks...

Why Does Android Use XML? Isn't That Inefficient?

Android is optimized for mobile devices with limited memory and horsepower, so you may find it strange that it uses XML so pervasively. After all, XML is a verbose, human-readable format not known for its brevity or efficiency, right?

Although you see XML when writing your program, the Eclipse plug-in invokes the Android resource compiler, aapt, to preprocess the XML into a compressed binary format. It is this format, not the original XML text, that is stored on the device.

A layout is a container for one or more child objects and a behavior to position them on the screen within the rectangle of the parent object. Here is a list of the most common layouts provided by Android:

- FrameLayout: classArranges its children so they all start at the top left of the screen. This is used for tabbed views and image switchers.
- LinearLayout: Arranges its children in a single column or row. This is the most common layout you will use.
- RelativeLayout: Arranges its children in relation to each other or to the parent. This is often used in forms.
- TableLayout: Arranges its children in rows and columns, similar to an HTML table.

Some parameters are common to all layouts:

xmlns:android="http://schemas.android.com/apk/res/android"
: Defines the XML namespace for Android. You should define this once, on the first XML tag in the file.

android:layout_width="fill_parent", android:layout_height="fill_parent"
: Takes up the entire width and height of the parent (in this case, the window). Possible values are fill_parent and wrap_content.

Inside the <*LinearLayout*> tag you'll find one child widget:

```
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello" />
```

This defines a simple text label. Let's replace that with some different text and a few buttons. Here's our first attempt:

Download Sudokuv1/res/layout/main1.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">
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/main_title" />
   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/continue_label" />
   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/new_game_label" />
   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/about_label" />
   <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/exit_label" />
</LinearLayout>
```

If you see warnings in the editor about missing grammar constraints (DTD or XML schema), just ignore them. Instead of hard-coding English text into the layout file, we use the @string/*resid* syntax to refer to strings in the res/values/strings.xml file. You can have different versions of this and other resource files based on the locale or other parameters such as screen resolution and orientation. Open that file now, switch to the strings.xml tab at the bottom if necessary, and enter this:

Download Sudokuv1/res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">Sudoku</string>
   <string name="main_title">Android Sudoku</string>
   <string name="continue_label">Continue</string>
   <string name="new_game_label">New Game</string>
   <string name="about_label">About</string>
   <string name="exit_label">Exit</string>
</resources>
```

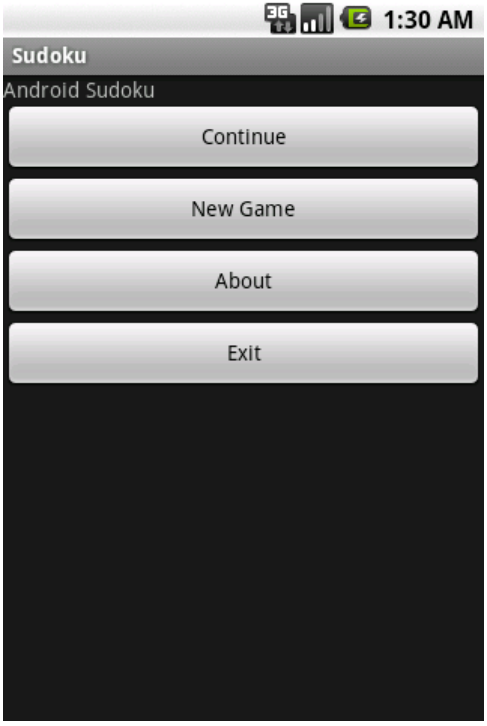

Figure 3.3: First version of the opening screen

Save strings.xml so Eclipse will rebuild the project. When you run the program now, you should see something like Figure 3.3. It's readable, but it could use some cosmetic changes.

Let's make the title text larger and centered, make the buttons smaller, and use a different background color. Here's the color definition, which you should put in res/values/colors.xml:

Download Sudokuv1/res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="background">#3500ffff</color>
</resources>
```

And here's the new layout:

Download Sudokuv1/res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:background="@color/background"
   android:layout_height="fill_parent"
   android:layout_width="fill_parent"
   android:padding="30dip"
   android:orientation="horizontal">
   <LinearLayout
      android:orientation="vertical"
      android:layout_height="wrap_content"
      android:layout_width="fill_parent"
      android:layout_gravity="center">
      <TextView
         android:text="@string/main_title"
         android:layout_height="wrap_content"
         android:layout_width="wrap_content"
         android:layout_gravity="center"
         android:layout_marginBottom="25dip"
         android:textSize="24.5sp" />
      <Button
         android:id="@+id/continue_button"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="@string/continue_label" />
      <Button
         android:id="@+id/new_button"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="@string/new_game_label" />
      <Button
         android:id="@+id/about_button"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="@string/about_label" />
      <Button
         android:id="@+id/exit_button"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="@string/exit_label" />
   </LinearLayout>
</LinearLayout>
```

In this version, we introduce a new syntax, @+id/*resid*. Instead of referring to a resource ID defined somewhere else, this is how you create a new resource ID to which others can refer. For example, @+id/about_button defines the ID for the About button, which we'll use later to make something happen when the user presses that button.

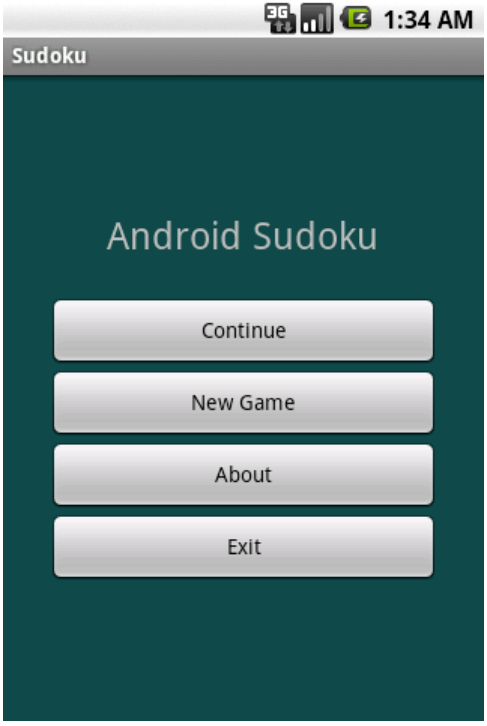

Figure 3.4: Opening screen with new layout

The result is shown in Figure 3.4. This new screen looks good in portrait mode (when the screen is taller than it is wide), but how about landscape mode (wide-screen)? The user can switch modes at any time, for example, by flipping out the keyboard or turning the phone on its side, so you need to handle that.

3.4 Using Alternate Resources

As a test, try switching the emulator to landscape mode (Ctrl+F11 or the 7 or 9 key on the keypad). Oops! The Exit button runs off the bottom of the screen (see Figure 3.5, on page 55). How do we fix that?

You could try to adjust the layout so that it works with all orientations. Unfortunately, that's often not possible or leads to odd-looking screens. When that happens, you'll need to create a different layout for landscape mode. That's the approach we'll take here.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Hello Android, Second Edition's Home Page
http://pragprog.com/titles/eband2
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/eband2.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)