Joseph Annuzzi, Jr.
Lauren Darcey
Shane Conder

Fifth Edition

Covers Android 6

Introduction to Android™ Application Development

Android Essentials

Praise for *Introduction to Android™ Application Development, Fifth Edition*

"*Introduction to Android Application Development* is a great resource for developers who want to understand Android app development but who have little or no experience with mobile software. This fifth edition has a bunch of great changes, from using Android Studio to understanding and implementing navigation patterns, and each chapter has quiz questions to help make sure you're picking up the vital info that fills this book."
—*Ian G. Clifton, author of* Android User Interface Design

"Revamped, revitalized, and refreshed! *Introduction to Android Application Development, Fifth Edition,* is a wonderful upgrade to an already impressive compendium. Common pitfalls are explained, new features are covered in depth, and the knowledge that the book is geared to cover everything from introduction of a concept to learning how to implement it into your app makes this a great choice for new developers who are ready to make the jump into Android development. Being already accustomed to the professional work and experience that Annuzzi et al., bring to the table, you will be grateful to have expert insight along with the care and instruction that developers of all skill levels can benefit from."
—*Phil Dutson, solution architect, ICON Health & Fitness*

"Best technical summary of Material Design implementation I've seen outside the Android sample docs."
—*Ray Rischpater, software development manager, Uber*

"*Introduction to Android Application Development* is well written and fulfills the requirements of developers, project managers, educators, and entrepreneurs in developing fully featured Android applications. In addition, it emphasizes quality assurance for mobile applications, teaches you how to design and plan your Android application, and teaches the software development process through a step-by-step, easy-to-understand approach. I recommend this book to anyone who wants to not just focus on developing apps, but also to apply tips and tricks and other tools for project management in their development of successful applications."
—*Bintu Harwani, founder of MCE (Microchip Computer Education)*

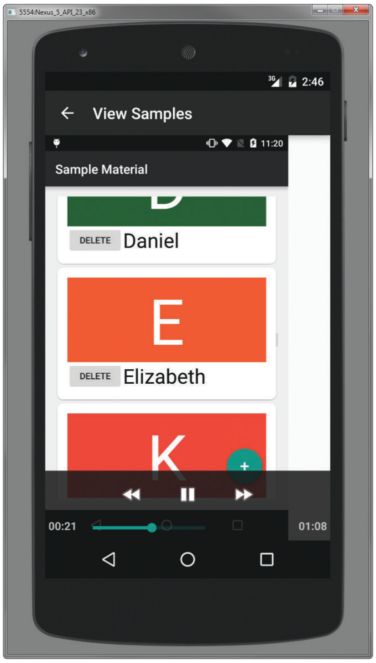

Figure 7.10 A `VideoView` playing a recording of a user interacting with an Application.

Here is an example XML layout resource definition for a `VideoView` control:

```
<VideoView
    android:id="@+id/video_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

And here is the `onCreate()` method of the `Activity`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_simple_video_view);
    VideoView vv = (VideoView) findViewById(R.id.videoView);
    MediaController mc = new MediaController(this);
    Uri video = Uri.parse("http://andys-veggie-garden.appspot.com/vid/reveal.
mp4");
    vv.setMediaController(mc);
    vv.setVideoURI(video);
}
```

You first want to get the `VideoView` from the layout, then you need to create a `MediaController` object. In our case, we are grabbing the video from the Internet so we first need to parse the URL of the video using the `Uri.parse` method so that our code uses a valid `Uri` object. We then use the `setMediaController()` method for adding the `MediaController` object to your `VideoView`, and then we use the `setVideoURI()` method to pass the `Uri` to our `VideoView`.

Since this example pulls the video from a location on the Internet, be sure to add the `INTERNET` permission of your Android manifest file as shown here:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Summary

The Android SDK provides many useful user interface components that developers can use to create compelling and easy-to-use applications. This chapter introduced you to many of the most useful controls and discussed how each behaves, how to style them, and how to handle input events from the user.

You learned how controls can be combined to create user entry forms. Important controls for forms include `EditText`, `Spinner`, and various `Button` controls. You also learned about controls that can indicate progress or the passage of time to users. We talked about many common user interface controls in this chapter; however, there are many others. In the next chapter, you will learn how to use various layout and container controls to organize a variety of on-screen controls easily and accurately.

Quiz Questions

1. What `Activity` method would you use to retrieve a `TextView` object?
2. What `TextView` method would you use to retrieve the text of that particular object?
3. What user interface control is used for retrieving text input from users?
4. What are the two different types of autocompletion controls?
5. True or false: A `Switch` control has three or more possible states.
6. True or false: The `DateView` control is used for retrieving dates from users.

Exercises

1. Create a simple application that accepts text input from a user with an `EditText` object and, when the user clicks an update `Button`, displays the text within a `TextView` control.
2. Create a simple application that has an integer defined in an integer resource file and, when the application is launched, displays the integer within a `TextView` control.

3. Create a simple application with a red color value defined in a color resource file. Define a `Button` control in the layout with a default blue `textColor` attribute. When the application is launched, have the default blue `textColor` value change to the red color value you defined in the color resource file.

References and More Information

Android API Guides: "User Interface":
http://d.android.com/guide/topics/ui/index.html
Android SDK Reference regarding the application `View` class:
http://d.android.com/reference/android/view/View.html
Android SDK Reference regarding the application `TextView` class:
http://d.android.com/reference/android/widget/TextView.html
Android SDK Reference regarding the application `EditText` class:
http://d.android.com/reference/android/widget/EditText.html
Android SDK Reference regarding the application `Button` class:
http://d.android.com/reference/android/widget/Button.html
Android SDK Reference regarding the application `CheckBox` class:
http://d.android.com/reference/android/widget/CheckBox.html
Android SDK Reference regarding the application `Switch` class:
http://d.android.com/reference/android/widget/Switch.html
Android SDK Reference regarding the application `RadioGroup` class:
http://d.android.com/reference/android/widget/RadioGroup.html
Android SDK Reference regarding the support v7 `Toolbar` class:
http://d.android.com/reference/android/support/v7/widget/Toolbar.html
Android SDK Reference regarding the application `VideoView` class:
http://d.android.com/reference/android/widget/VideoView.html

8 Positioning with Layouts

In this chapter, we discuss how to design user interfaces for Android applications. Here, we focus on the various layout controls you can use to organize screen elements in different ways. We also cover some of the more complex `View` controls that we call container views. These are `View` controls that can contain other `View` controls.

Creating User Interfaces in Android

Application user interfaces can be simple or complex, involving many different screens or only a few. Layouts and user interface controls can be defined as application resources or created programmatically at runtime.

Although it's a bit confusing, the term *layout* is used for two different but related purposes in Android user interface design:

- In terms of resources, the `res/layout/` directory contains XML resource definitions often called layout resource files. These XML files provide a template for how to arrange and draw controls on the screen; layout resource files may contain any number of controls.
- The term is also used to refer to a set of `ViewGroup` classes, such as `LinearLayout`, `FrameLayout`, `TableLayout`, `RelativeLayout`, and `GridLayout`. These controls are used to organize other `View` controls. We talk more about these classes later in this chapter.

Creating Layouts Using XML Resources

As discussed in previous chapters, Android provides a simple way to create layout resource files in XML. These resources are stored in `res/layout/`. This is the most common and convenient way to build Android user interfaces and is especially useful for defining screen elements and default control properties that you know about at compile time. These layout resources are then used much like templates. They are loaded with default attributes that you can modify programmatically at runtime.

You can configure almost any `View` or `ViewGroup` subclass attribute using the XML layout resource files. This method greatly simplifies the user interface design process, moving much of the static creation and layout of user interface controls, and basic definition of control attributes, to the XML instead of littering the code. Developers reserve the ability to alter these layouts programmatically as necessary, but they should set all the defaults in the XML template whenever possible.

You'll recognize the following as a simple layout file with a `RelativeLayout` and a single `TextView` control. Here is the default layout file provided with any new Android project in Android Studio, referred to as `res/layout/activity_main.xml`, assuming your `Activity` is named `MainActivity`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

This block of XML shows a basic layout with a single `TextView` control. The first line, which you might recognize from most XML files, is required with the `android` layout namespace, as shown. Because it's common across all the files, we do not show it in any other examples.

Next, we have the `RelativeLayout` element. `RelativeLayout` is a `ViewGroup` that shows each child `View` relative to other views. When applied to a full screen, it merely means that each child `View` is drawn relative to the specified `View`.

Finally, there is a single child `View`—in this case, a `TextView`. A `TextView` is a control that is also a `View`. A `TextView` draws text on the screen. In this case, it draws the text defined in the `"@string/hello"` string resource.

If you create your own XML file, though, that file won't actually draw anything on the screen. A particular layout is usually associated with a particular `Activity`. In your default Android project, there is only one `Activity`, which sets the `activity_main.xml` layout by default. To associate the `activity_main.xml` layout with the `Activity`, use the method call `setContentView()` with the identifier of the `activity_main.xml` layout.

The ID of the layout matches the XML filename without the extension. In this case, the preceding example came from `activity_main.xml`, so the identifier of this layout is simply `activity_main.xml`, and this layout will actually display on the screen as it has been created for us during project creation:

```
setContentView(R.layout.activity_main);
```

Warning

The Android tools team has made every effort to make the Android Studio `Design` view layout editor feature complete, and this tool can be helpful for designing and previewing how layout resources will look on a variety of different devices. However, the preview can't replicate exactly how the layout appears to end users. For this, you must test your application on a properly configured emulator and, more important, on your target devices.

Creating Layouts Programmatically

You can create user interface components such as layouts at runtime programmatically, but for organization and maintainability, it's best to leave this for the odd case rather than the norm. The main reason is that the creation of layouts programmatically is onerous and difficult to maintain, whereas the XML resources are visual and more organized, and could be used by a separate designer with no Java skills.

Tip

The code examples provided in this section are taken from the `SameLayout` application. The source code for the `SameLayout` application is provided for download on the book's website (*http://introductiontoandroid.blogspot.com*).

The following example shows how to have an `Activity` instantiate a `LinearLayout` programmatically and place two `TextView` controls within it as child controls. The same two string resources are used for the contents of the controls; these actions are done at runtime instead.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    assert getSupportActionBar() != null;
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    TextView text1 = new TextView(this);
    text1.setText(R.string.string1);

    TextView text2 = new TextView(this);
```

(*Continues*)