21. Manual XML Layout Design in Android Studio

While the design of layouts using the Android Studio Layout Editor tool greatly improves productivity, it is still possible to create XML layouts by manually editing the underlying XML. This chapter will introduce the basics of the Android XML layout file format.

21.1 Manually Creating an XML Layout

The structure of an XML layout file is actually quite straightforward and follows the hierarchical approach of the view tree. The first line of an XML resource file should ideally include the following standard declaration:

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

This declaration should be followed by the root element of the layout, typically a container view such as a layout manager. This is represented by both opening and closing tags and any properties that need to be set on the view. The following XML, for example, declares a ConstraintLayout view as the root element, assigns the ID *activity_main* and sets *match_parent* attributes such that it fills all the available space of the device display:

```
<android.support.constraint.ConstraintLayout xmlns:android="http://scl
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main_activity"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.ebookfrenzy.demoapp.MainActivity">

</android.support.constraint.ConstraintLayout>
```

Note that in the above example the layout element is also configured with padding on each side of 16dp (density independent pixels). Any specification of spacing in an Android layout must be specified using one of the following units of measurement:

- **in** – Inches.
- **mm** – Millimeters.
- **pt** – Points (1/72 of an inch).
- **dp** – Density-independent pixels. An abstract unit of measurement based on the physical density of the device display relative to a 160dpi display baseline.
- **sp** – Scale-independent pixels. Similar to dp but scaled based on the user's font preference.
- **px** – Actual screen pixels. Use is not recommended since different displays will have different pixels per inch. Use *dp* in preference to this unit.

Any children that need to be added to the ConstraintLayout parent must be *nested* within the opening and closing tags. In the following example a Button widget has been added as a child of the ConstraintLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://sch
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.ebookfrenzy.demoapp.MainActivity">

    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button" />

</android.support.constraint.ConstraintLayout>
```

As currently implemented, the button has no constraint connections. At runtime, therefore, the button will appear in the top left-hand corner of the screen (though indented 16dp by the padding assigned to the parent layout).

If opposing constraints are added to the sides of the button, however, it will appear centered within the layout:

```
<Button
    android:text="Button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    app:layout_constraintLeft_toLeftOf="@+id/activity_main"
    app:layout_constraintTop_toTopOf="@+id/activity_main"
    app:layout_constraintRight_toRightOf="@+id/activity_main"
    app:layout_constraintBottom_toBottomOf="@+id/activity_main" />
```

Note that each of the constraints is attached to the element named *activity_main* which is, in this case, the parent ConstraintLayout instance.

To add a second widget to the layout, simply embed it within the body of ConstraintLayout element. The following modification, for example, adds a TextView widget to the layout:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.ebookfrenzy.demoapp.MainActivity">

    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        app:layout_constraintLeft_toLeftOf="@+id/activity_main"
        app:layout_constraintTop_toTopOf="@+id/activity_main"
        app:layout_constraintRight_toRightOf="@+id/activity_main"
        app:layout_constraintBottom_toBottomOf="@+id/activity_main" /

    <TextView
```

```
        android:text="TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView" />

</android.support.constraint.ConstraintLayout>
```

Once again, the absence of constraints on the newly added TextView will cause it to appear in the top left-hand corner of the layout at runtime. The following modifications add opposing constraints connected to the parent layout to center the widget horizontally, together with a constraint connecting the bottom of the TextView to the top of the button with a margin of 72dp:

```
<TextView
    android:text="TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    app:layout_constraintLeft_toLeftOf="@+id/activity_main"
    app:layout_constraintRight_toRightOf="@+id/activity_main"
    app:layout_constraintBottom_toTopOf="@+id/button"
    android:layout_marginBottom="72dp" />
```

Also, note that the Button and TextView views have a number of attributes declared. Both views have been assigned IDs and configured to display text strings represented by string resources named *button_string* and *text_string* respectively. Additionally, the *wrap_content* height and width properties have been declared on both objects so that they are sized to accommodate the content (in this case the text referenced by the string resource value).

Viewed from within the Preview panel of the Layout Editor in Text mode, the above layout will be rendered as shown in Figure 21-1:

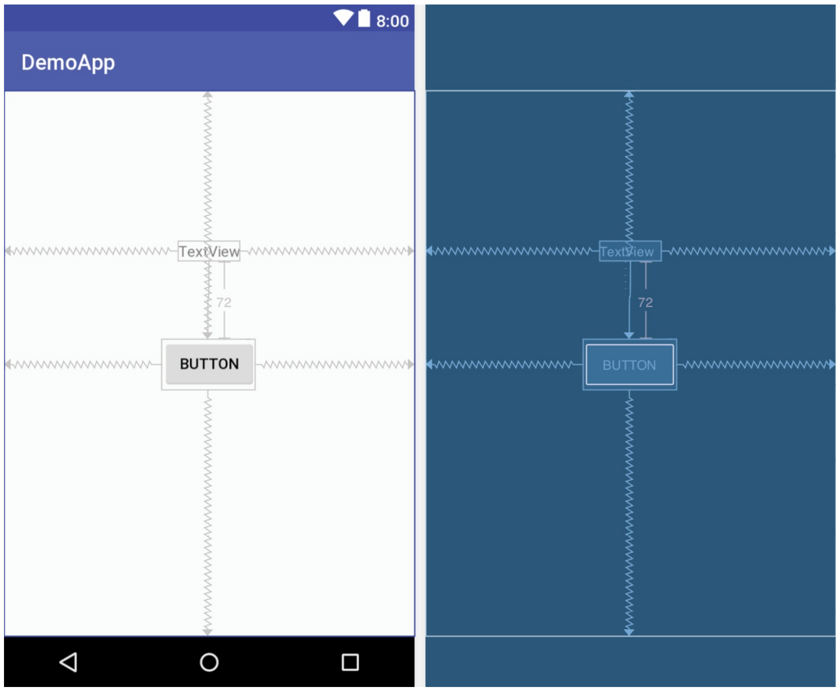

Figure 21-1

21.2 Manual XML vs. Visual Layout Design

When to write XML manually as opposed to using the Layout Editor tool in design mode is a matter of personal preference. There are, however, advantages to using design mode.

First, design mode will generally be quicker given that it avoids the necessity to type lines of XML. Additionally, design mode avoids the need to learn the intricacies of the various property values of the Android SDK view classes. Rather than continually refer to the Android documentation to find the correct keywords and values, most properties can be located by referring to the Attributes panel.

All the advantages of design mode aside, it is important to keep in mind that the two approaches to user interface design are in no way mutually exclusive. As an application developer, it is quite likely that you will end up creating user interfaces within design mode while performing fine-tuning and layout tweaks of the design by directly editing the generated XML resources. Both views of the interface design are, after all, displayed side by side within the Android Studio environment making it easy to work seamlessly on both the XML and the visual layout.

21.3 Summary

The Android Studio Layout Editor tool provides a visually intuitive method for designing user interfaces. Using a drag and drop paradigm combined with a set of property editors, the tool provides considerable productivity benefits to the application developer.

User interface designs may also be implemented by manually writing the XML layout resource files, the format of which is well structured and easily understood.

The fact that the Layout Editor tool generates XML resource files means that these two approaches to interface design can be combined to provide a “best of both worlds” approach to user interface development.