An Android Studio TableLayout and TableRow Tutorial

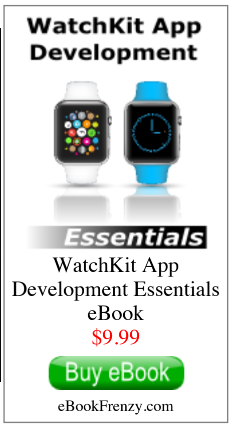
Previous
An Overview of Android SQLite Databases in Android Studio

Table of Contents

Next
An Android Studio SQLite Database Tutorial

When the work began on the next chapter of this book (An Android Studio SQLite Database Tutorial) it was originally intended that it would include the steps to design the user interface layout for the database example application. It quickly became evident, however, that the best way to implement the user interface was to make use of the Android TableLayout and TableRow views and that this topic area deserved a self-contained chapter. As a result, this chapter will focus solely on the user interface design of the database application completed in the next chapter, and in doing so, take some time to introduce the basic concepts of table layouts in Android Studio.

Contents



The TableLayout and TableRow Layout Views

The purpose of the TableLayout container view is to allow user interface elements to be organized on the screen in a table format consisting of rows and columns. Each row within a TableLayout is occupied by a TableRow instance, which, in turn, is divided into cells, with each cell containing a single child view (which may itself be a container with multiple view children).

The number of columns in a table is dictated by the row with the most columns and, by default, the width of each column is defined by the widest cell in that column. Columns may be configured to be shrinkable or stretchable (or both) such that they change in size relative to the parent TableLayout. In addition, a single cell may be configured to span multiple columns.

Consider the user interface layout shown in Figure 41-1:

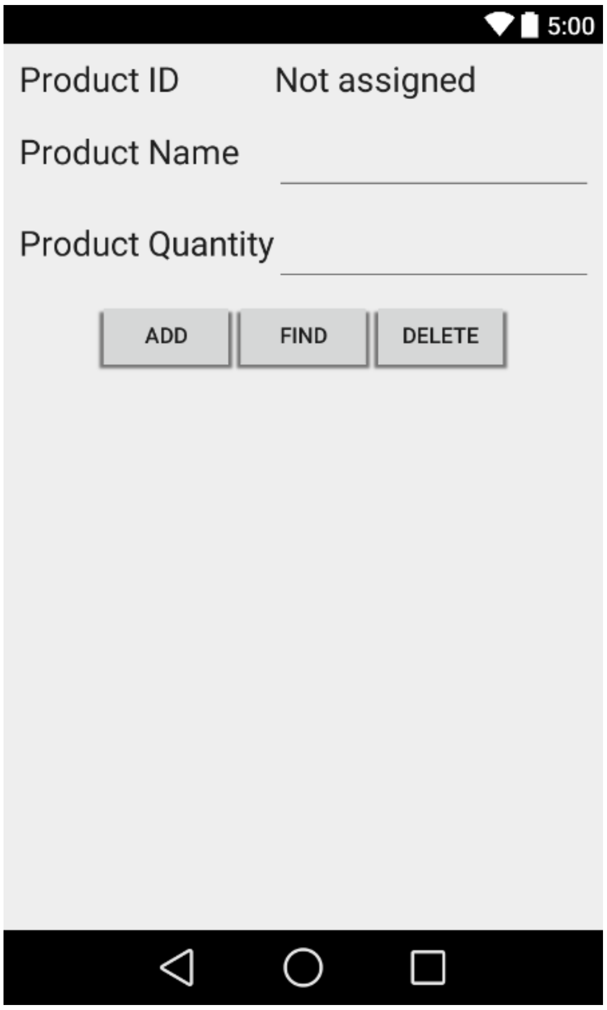


Figure 41-1

From the visual appearance of the layout, it is difficult to identify the TableLayout structure used to design the interface. The hierarchical tree illustrated in Figure 41-2, however, makes the structure a little easier to understand:

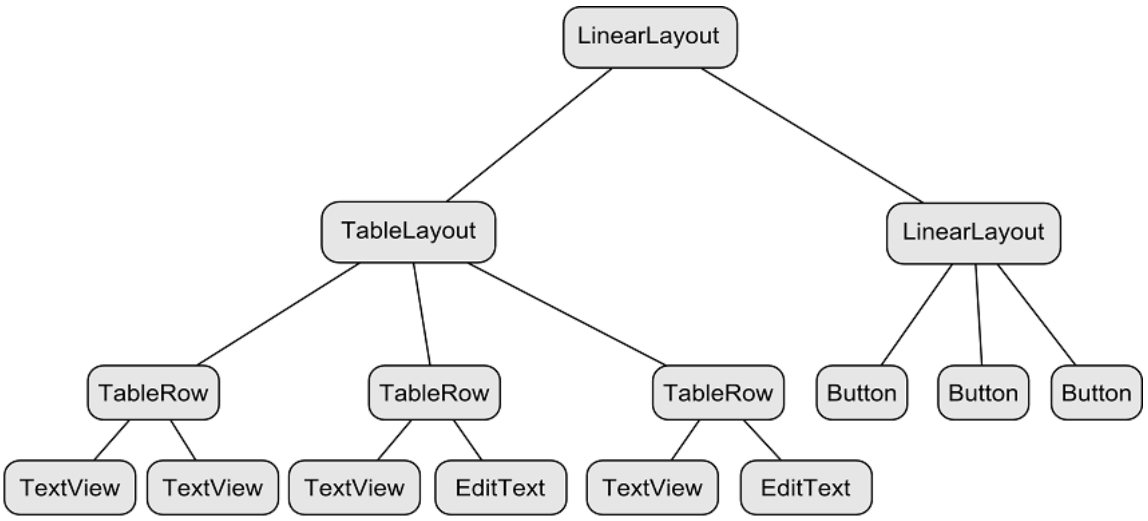


Figure 41-2

Clearly, the layout consists of a parent LinearLayout view with TableLayout and LinearLayout children. The TableLayout contains three TableRow children representing three rows in the table. The TableRows contain two child views, with each child representing the contents of a column cell. The LinearLayout child view contains three Button children.

The layout shown in Figure 41-2 is the exact layout that is required for the database example that will be completed in the next chapter. The remainder of this chapter, therefore, will be used to work step by step through the design of this user interface using the Android Studio Designer tool.

Creating the Database Project

Start Android Studio and create a new project, entering Database into the Application name field and ebookfrenzy.com as the Company Domain setting before clicking on the Next button.

On the form factors screen, enable the Phone and Tablet option and set the minimum SDK setting to API 8: Android 2.2 (Froyo). Continue to proceed through the screens, requesting the creation of a blank activity named DatabaseActivity with corresponding layout and menu resource files named activity_database and menu_database respectively.

Adding the TableLayout to the User Interface

Locate the activity_database.xml file in the Project tool window (app -> res -> layout) and double click on it to load it into the Designer tool. By default, Android Studio has used a RelativeLayout as the root layout element in the user interface. This needs to be replaced by a vertically oriented LinearLayout. With the Designer tool in Text mode, replace the XML with the following:

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


</LinearLayout>
```

Switch to Design mode and, referring to the Layouts section of the Palette, drag and drop a TableLayout view so that it is positioned at the top of the LinearLayout canvas area as illustrated in Figure 41-3:

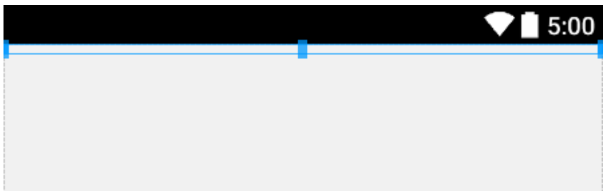


Figure 41-3

Once these initial steps are complete, the Component Tree for the layout should resemble that shown in Figure 41-4:

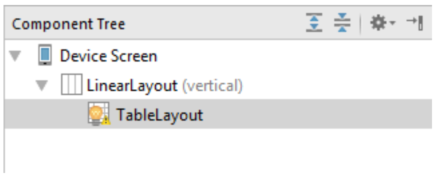


Figure 41-4

Adding and Configuring the TableRows

Now that the TableLayout has been added to the user interface layout, three TableRow instances need to be added as children. From the Designer palette, locate the TableRow entry listed under Layouts and drag and drop an instance directly onto the top of the TableLayout entry in the Component Tree panel. Repeat this step to add two more TableRows so that the component tree matches Figure 41-5:

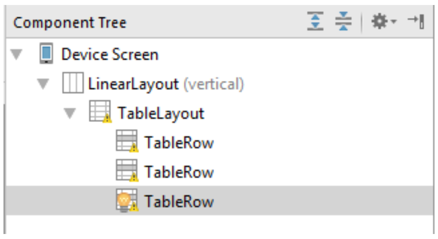


Figure 41-5

From within the Widgets section of the palette, drag and drop two Large Text TextView objects onto the uppermost TableRow entry in the Component Tree (Figure 41-6):

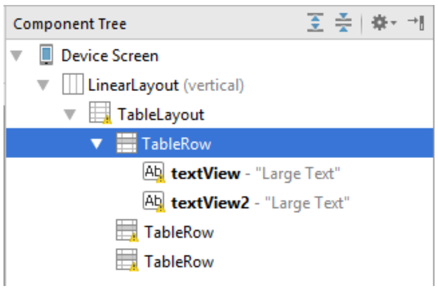


Figure 41-6

Double click on the left most TextView within the screen layout and, in the quick property settings panel, change the text property to “Product ID”. Repeat this step for the right most TextView, this time changing the text to “Not assigned” and specifying an id value of @+id/productID. Extract the text for each TextView to new string resources using the light bulb icon displayed when the view is selected.

Drag and drop another Large Text view and a Plain Text Edit Text view from the Text Fields section of the palette onto the second TableRow entry in the Component Tree . Change the text on the TextView to Product Name and the ID of the EditText object to @+id/productName.

Drag and drop another Plain Text view and a Number (Decimal) Text Field onto the third TableRow. Change the text on the TextView to Product Quantity and the ID of the Text Field object to @+id/productQuantity. Before proceeding, be sure to extract all of the text properties added in the above steps to string resources.

Adding the Button Bar to the Layout

The next step is to add a LinearLayout (Horizontal) view to the parent LinearLayout view, positioned immediately below the TableLayout view. Drag and drop a LinearLayout (Horizontal) instance from the Layouts section of the Designer palette and drop it directly onto the LinearLayout (Vertical) entry in the Component Tree panel.

Drag and drop three Button objects onto the new LinearLayout and assign string resources for each button that read “Add”, “Find” and “Delete” respectively.

With the new horizontal Linear Layout view selected in the Component Tree, click on the Gravity button in the Designer toolbar (Figure 41-7) and select the Center gravity option so that the buttons are centered horizontally within the display:

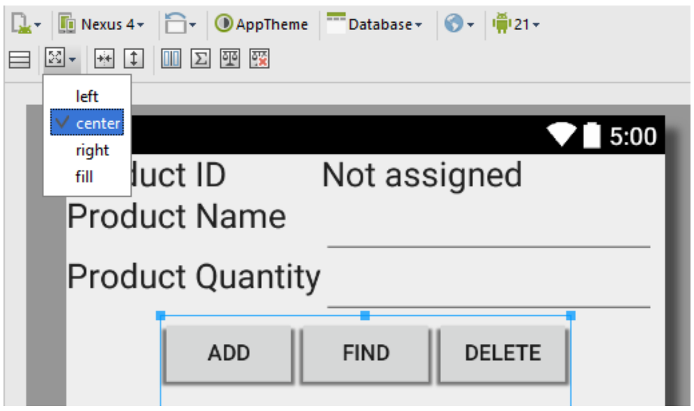
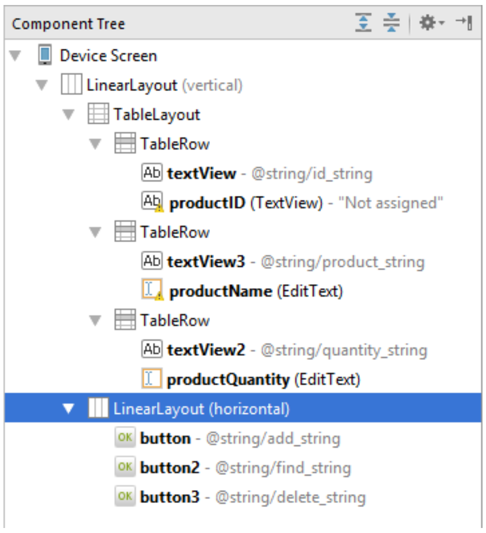


Figure 41-7

Before proceeding, also check the hierarchy of the layout in the Component Tree panel, taking extra care to ensure the view ID names match those in the following figure:

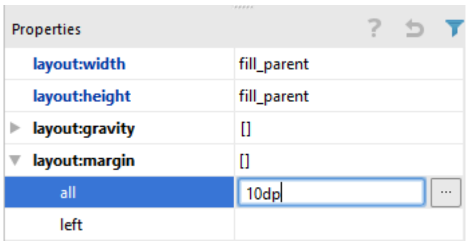


```
Component Tree
Device Screen
  LinearLayout (vertical)
    TableLayout
      TableRow
        textView - @string/id_string
        productID (TextView) - "Not assigned"
      TableRow
        textView3 - @string/product_string
        productName (EditText)
      TableRow
        textView2 - @string/quantity_string
        productQuantity (EditText)
    LinearLayout (horizontal)
      button - @string/add_string
      button2 - @string/find_string
      button3 - @string/delete_string
```

Figure 41-8

Adjusting the Layout Margins

All that remains is to adjust some of the layout settings. Begin by clicking on the first TableRow entry in the Component Tree panel so that it is selected. Hold down the Ctrl-key on the keyboard and click in the second and third TableRows so that all three items are selected. In the Properties panel, locate the layout:margin property category and, once located, unfold the category and change the all setting to 10dp as shown in Figure 41-9:



Properties	
layout:width	fill_parent
layout:height	fill_parent
layout:gravity	[]
layout:margin	[]
all	10dp
left	

Figure 41-9

With margins set on all four sides of the three TableRows, the user interface should appear as illustrated in Figure 41-1. For the sake of completeness, and for comparison purposes in the event that your layout does not match that in Figure 41-1, the full activity_database.xml structure for this layout is outlined below.

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

    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal">

        <TableRow
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_margin="10dp">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
                android:text="@string/id_string"
                android:id="@+id/textView" />

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
                android:text="Not assigned"
                android:id="@+id/productID" />
        </TableRow>

        <TableRow
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_margin="10dp">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
                android:text="@string/product_string"
                android:id="@+id/textView3" />

            <EditText
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/productName" />
        </TableRow>

        <TableRow
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_margin="10dp">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
                android:text="@string/quantity_string"
                android:id="@+id/textView2" />

            <EditText
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:inputType="numberDecimal"
                android:ems="10"
                android:id="@+id/productQuantity" />
        </TableRow>
    </TableLayout>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_gravity="center_horizontal"
        android:layout_margin="10dp">

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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/find_string"
            android:id="@+id/button2" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/delete_string"
            android:id="@+id/button3" />
    </LinearLayout>
```

```
</LinearLayout>
```

Summary

The Android TableLayout container view provides a way to arrange view components in a row and column configuration. Whilst the TableLayout view provides the overall container, each row, and the cells contained therein, are implemented via instances of the TableRow view. In this chapter, a user interface has been designed in Android Studio using the TableLayout and TableRow containers. The next chapter will add the functionality behind this user interface to implement the SQLite database capabilities.

Previous
An Overview of Android SQLite Databases in Android Studio

Table of Contents

Next
An Android Studio SQLite Database Tutorial

Retrieved from "http://www.techotopia.com/index.php/An_Android_Studio_TableLayout_and_TableRow_Tutorial"

- This page was last modified 18:18, 20 January 2015.
-