

## THE SCREEN LAYOUT AND THE MAIN.XML FILE CONTD...

- The RelativeLayout allows the programmer to position controls such as buttons on the screen relative to each other, such as above or below. To try a RelativeLayout, modify your main.xml to look like the following, and then restart your application:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:id="@+id/centertext"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello"
android:layout_centerHorizontal="true"
android:layout_centerVertical="true"
/>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Text View number two"
android:layout_above="@id/centertext"
/>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Text View number three"
android:layout_below="@id/centertext"/>
</RelativeLayout>
```

- Notice that the second and third TextViews appear at the top and bottom, respectively.

- `android:id="@+id/centertext"`  
This is required because the other two `TextView` objects need to refer to the position of the first and need a way to identify it. The programmer can choose any ID for the control; it's a good idea to choose a short name that identifies the control.
- This prefix on the id name causes the actual identification value to be entered in that inner class. Take a look at the `R.java` file by doubleclicking the `R.java` entry in the Package Explorer window on the left side of the Screen and it opens in the editor. You will see the entry `centertext` in the `id` inner class.
- The `TableLayout` is similar in appearance to the Java Frame's `GridLayout`, but it is configured quite differently.
- Within the `TableLayout`, the programmer makes one or more `TableRow` entries.
- Controls are placed inside the `TableRow` and appear in the order in which they are entered into the XML file.
- Columns are created as entries are made; however, to skip a column, the programmer can make a zero-based column specification.
- If only one row is specified, no gap appears between columns, even if there is a gap specified in the column indexes.
- If multiple rows are used, gaps appear where column index values are skipped. The column index is specified by the `android:layout_column` attribute (`android:layout_column="1"`).
- Modify your `main.xml` file to match the following. Note that several lines necessary for the `RelativeLayout` have been removed:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TableRow>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Column 1"
android:layout_column="0"
/>
<TextView
```

```

android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Column 2"
android:layout_column="1"
/>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Column 3"
android:layout_column="3"
/>
</TableRow>
<TableRow>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Column 1"
android:layout_column="0"
/>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Column 2"
android:layout_column="2"
/>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Column 3"
android:layout_column="4"
/>
</TableRow>
</TableLayout>

```

- When you rerun the application, your emulator should look like the given figure.
- Remember, we are actually using five columns, indexed zero through four (0,1,2,3,4), and they are specified by the attribute `android:layout_column`. They have nothing to do with the attribute `android:text`, although we are using the word Column in the text.



- It is common for the Android programmer to put one layout manager inside another to achieve a desired screen layout.
- In the above example, the outermost layout is a LinearLayout, and it contains two TableLayouts. Each of the TableLayouts contains a single row, with several TextViews included in the row.
- A programmer might do this to produce rows with differing numbers of columns, or maybe two groups with rows and columns, but containing some other object or layout of many objects in the middle.
- Example:

```
<?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"
```

```
>
```

```
<TableLayout
```

```
  android:background="#0000ff"
```

```
  android:layout_width="fill_parent"
```

```
  android:layout_height="wrap_content"
```

```
>
```

```
<TableRow>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"

android:text="Column 1"
android:layout_column="0"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Column 2"
android:layout_column="1"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Column 3"
android:layout_column="3"
/>
</TableRow>
</TableLayout>
<TableLayout
android:background="#ff0000"
android:layout_width="fill_parent"
android:layout_height="wrap_content">
<TableRow>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Column 1"
android:layout_column="0"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Column 2"
android:layout_column="2"
/>
<TextView
android:layout_width="wrap_content"
```

```

android:layout_height="wrap_content"
android:text="Column 3"
android:layout_column="4"
/>
</TableRow>
</TableLayout>
</LinearLayout>

```

- The orientation attribute in the LinearLayout is changed to vertical. This causes the multiple embedded layouts to be stacked top to bottom instead of left to right. If the orientation value is left as horizontal, the Linear-Layout would attempt to place the embedded TableLayouts side by side, which is not the intent here.
- Second, notice that we added background attributes to the TableLayouts; in this case, we set them to a color. The first one is solid blue(#0000FF) and the second is solid red (#FF0000).
- Colors are made of three component values: one for red, one for green, and one for blue. Each component can have a value ranging from 0 to 255. The values are specified in hexadecimal, or base-16. The hexadecimal values and their decimal equivalents are specified in Table 3.2.

**Table 3.2 Hexadecimal Values and Their Decimal Equivalents**

Hex Value	Decimal Equivalent
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

- The hexadecimal equivalent for 255 is FF (15\*16 + 15). Next to the pound sign (#) are three pairs of hex values, one for each color. So in our sample values, #FF0000 gives the maximum value for red (FF or 255), and zero for both green and blue. This yields the brightest, purest red. Likewise, #0000FF gives us the brightest, purest blue. All zeroes give us black, and all Fs give us white. In any case, where we use the same value for each red, green, and blue, we have a shade of gray. The higher the numbers, the lighter the shade. Combinations yield different colors.
- Table 3.3 shows some combinations for common colors.

**Table 3.3 RGB Values in Hexadecimal and Their Color Name Equivalents**

<b>RGB Value</b>	<b>Corresponding Color</b>
#FFFF00	Yellow
#FF00FF	Magenta
#00FFFF	Cyan
#FFA500	Orange
#A0522D	Sienna

- We added colors here so you could see where one TableLayout ends and the other begins. There are several ways to add background colors to Android controls and layouts. The one used here is crude, but it is simple and does the trick. Load this as your main.xml file and restart your application. Figure 3.4 shows what you should get in your emulator.
- Pay particular attention to two attributes in any of the layouts—android:layout\_width and android:layout\_height—and their typical settings: either fill\_parent or wrap\_content.
- If you want a layout component to stretch all the way from left to right, set android:layout\_width to fill\_parent; if not, set it to wrap\_content.
- The same goes for top to bottom. If you want to cover the whole screen, set android:layout\_height to fill\_parent; if not, set it to wrap\_content.
- If you expect to share a particular direction with more than one component or layout, you will use wrap\_content for that direction.



- Note how the two TableLayouts have wrap\_content as their android:layout\_height setting because they are expected to fit on top of each other. Experiment with these on your own.
- Typically, the outermost layout has both of these set to fill\_parent.

### Creating Font objects.

- Following is an example:  

```
Font myfont=new Font("SansSerif", Font.BOLD, 32);
```
- The example specifies a type name, a typestyle (bold), and a point size. These, along with color, are typical settings you would want to control for your application's text.
- Of course, you can use different styles for different text on the application. For each layout or control, you can make the following specifications regarding text:

android:textColor  
 android:textAppearance  
 android:textSize  
 android:typeface  
 android:textStyle

- You would enter these specifications, with allowable values of course, just as you would with the other attributes in the main.xml file. However, the more variations you want to use, the more tedious entering these attributes can become.

- There is another option. You can create some standard combinations, give them names, and refer to them as you need them.
- Table 3.4 shows a brief list of common files, suggested filenames, their resource types, and the ideal place to put them in the project. Filenames must be lowercase and simple, containing numbers, letters, and underscores only.

**Table 3.4 Resource File Types and Details**

Resource Type	Directory Location	Suggested Filename	XML Tag, if Applicable
Strings	/res/values	strings.xml	<string>
Arrays of strings	/res/values	arrays.xml	<string-array>
Color values	/res/values	colors.xml	<color>
Bitmap graphics	/res/drawable	Any with .png, .jpg, or .gif extensions	n/a
Menu files	/res/menu	menu1.xml	<menu>
XML files (special purpose not described elsewhere)	/res/xml	special.xml	Defined by programmer
Raw files (for example, text files)	/res/raw	Any with .txt, .mp3, and so on extensions	n/a
Layout files	/res/layout (created by Eclipse)	start.xml, secondary.xml	n/a
Styles and themes	/res/values	textstyles.xml	<style>

- **Example:** To create a bolder type style we will call shout to emphasize particular text items. We want to use red for the text, make it a bigger size than the default, and use bold print to add even more emphasis. We see that files controlling text styles belong in the /res/values directory.
- In the Package Explorer window, right-click the /res/values folder and select New from the menu; then select File from the list. Name the file textstyles.xml or something similar of your own choosing, and enter the following:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="shout">
<item name="android:textColor">#ff0000</item>
<item name="android:textSize">20pt</item>
<item name="android:textStyle">bold</item>
</style>
</resources>
```

- Notice that in the name-value pairs, the names are the attributes you would use in the main.xml file if you chose to create the special effects there. The same holds true for the values. Be careful to properly form the XML file. Now, whenever you want to use this configuration for emphasized text, simply add the following line to the item in the main.xml file (or any other layout file for that matter).  
style="@style/shout"
- Test your special effects by adding this line to the first TextView entry in the main.xml file. The first TextView entry in the main.xml file should look like this:  

```
<TextView
style="@style/shout"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Column 1"
android:layout_column="0"
/>
```
- When you restart the application, the results should look like Figure 3.5 in the emulator. Try adding some other specification combinations in your styles.xml file, apply them to other controls or layouts in your main.xml file as we did earlier, and examine the results.
- Remember that name-value pairs in the styles file should be appropriate for the control or layout you attempt to apply them to; for instance, type styles should be applied to items that have text as a feature.



For any doubt contact 9873961590.