Module 3 : UI Design and Data storage

UNITS OF MEASUREMENT

When specifying the size of an element on an Android UI, you should be aware of the following units of measurement:

- `dp` — Density-independent pixel. 160dp is equivalent to one inch of physical screen size. This is the recommended unit of measurement when specifying the dimension of views in your layout. You can specify either "dp" or "dip" when referring to a density-independent pixel.
- `sp` — Scale-independent pixel. This is similar to `dp` and is recommended for specifying font sizes.
- `pt` — Point. A point is defined to be 1/72 of an inch, based on the physical screen size.
- `px` — Pixel. Corresponds to actual pixels on the screen. Using this unit is not recommended, as your UI may not render correctly on devices with different screen sizes.

Here, you set the width of both the TextView and Button views to an absolute value. In this case, the width for the TextView is set to 105 density-independent pixels wide, and the Button to 160 density independent pixels wide. Figure 3-1 shows how the views look when viewed on an emulator with a resolution of 320×480. Figure 3-2 shows how the views look when viewed on a high-resolution (480×800) emulator.

Figure 3-1 **Figure 3-2**

As you can see, in both emulators the widths of both views are the same with respect to the width of the emulator. This demonstrates the usefulness of using the dp unit, which ensures that even if the resolution of the target device is different, the size of the view relative to the device remains unchanged. The preceding example also specifies that the orientation of the layout is vertical:

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

The default orientation layout is horizontal, so if you omit the android:orientation attribute, the views will appear as shown in Figure 3-3.

Figure 3-3

In LinearLayout, you can apply the layout_weight and layout_gravity attributes to views contained within it, as the following modifications to main.xml show:

```
<?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="105dp"
android:layout_height="wrap_content"
android:text="@string/hello"
/>
<Button
android:layout_width="160dp"
android:layout_height="wrap_content"
android:text="Button"
```

```
android:layout_gravity="right"
android:layout_weight="0.2"
/>
<EditText
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"
android:layout_weight="0.8"
/>
</LinearLayout>
```

Figure 3-4 shows that the button is aligned to the right of its parent (which is the LinearLayout) using the layout_gravity attribute. At the same time, you use the layout_weight attribute to specify the ratio in which the Button and EditText views occupy the remaining space on the screen. The total value for the layout_weight attribute must be equal to 1.

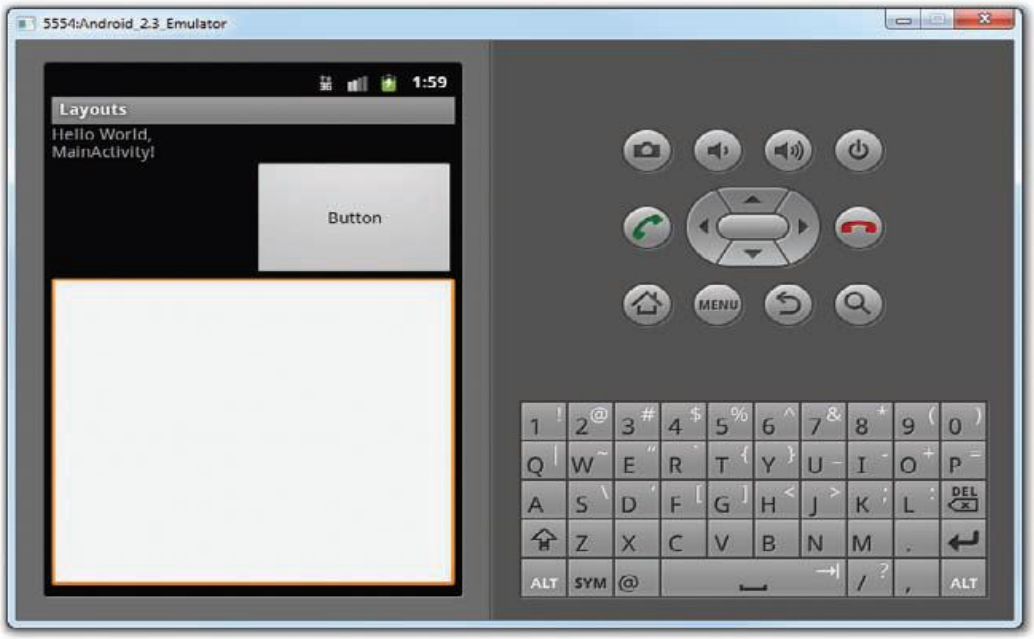

Figure 3-4

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. Consider the following UI defined in main.xml:

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

```
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
xmlns:android=http://schemas.android.com/apk/res/android

>
<Button
android:layout_width=”188dp”
android:layout_height=”wrap_content”
android:text=”Button”
android:layout_x=”126px”
android:layout_y=”361px”
/>
<Button
android:layout_width=”113dp”
android:layout_height=”wrap_content”
android:text=”Button”
android:layout_x=”12px”
android:layout_y=”361px”
/>
</AbsoluteLayout>
```

Figure 3-5 shows the two Button views located at their specified positions using the android_layout_x and android_layout_y attributes.

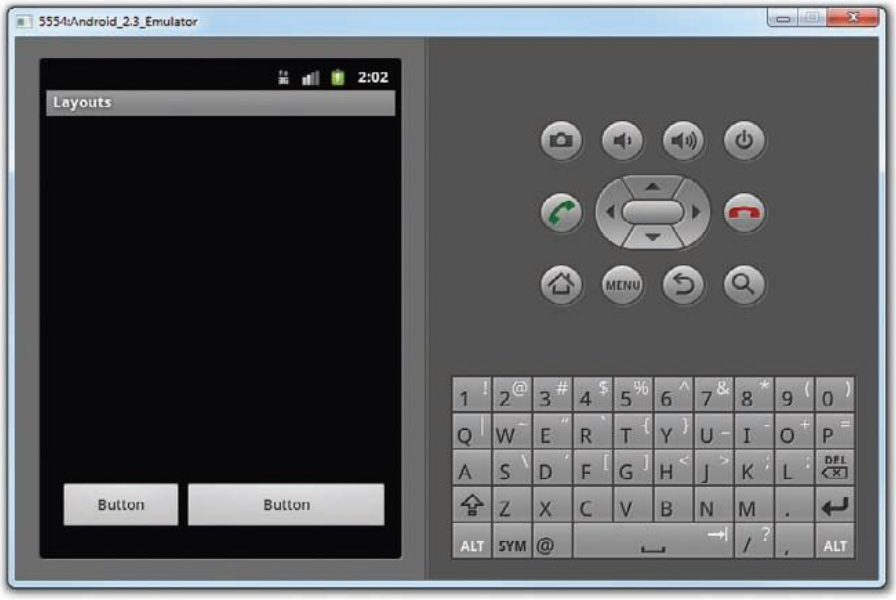

Figure 3-5

However, there is a problem with the AbsoluteLayout when the activity is viewed on a high-resolution screen (see Figure 3-6). For this reason, the AbsoluteLayout has been deprecated since Android 1.5 (although it is still supported in the current version). You should avoid using the AbsoluteLayout in your UI, as it is not guaranteed to be supported in future versions of Android. You should instead use the other layouts described in this chapter.

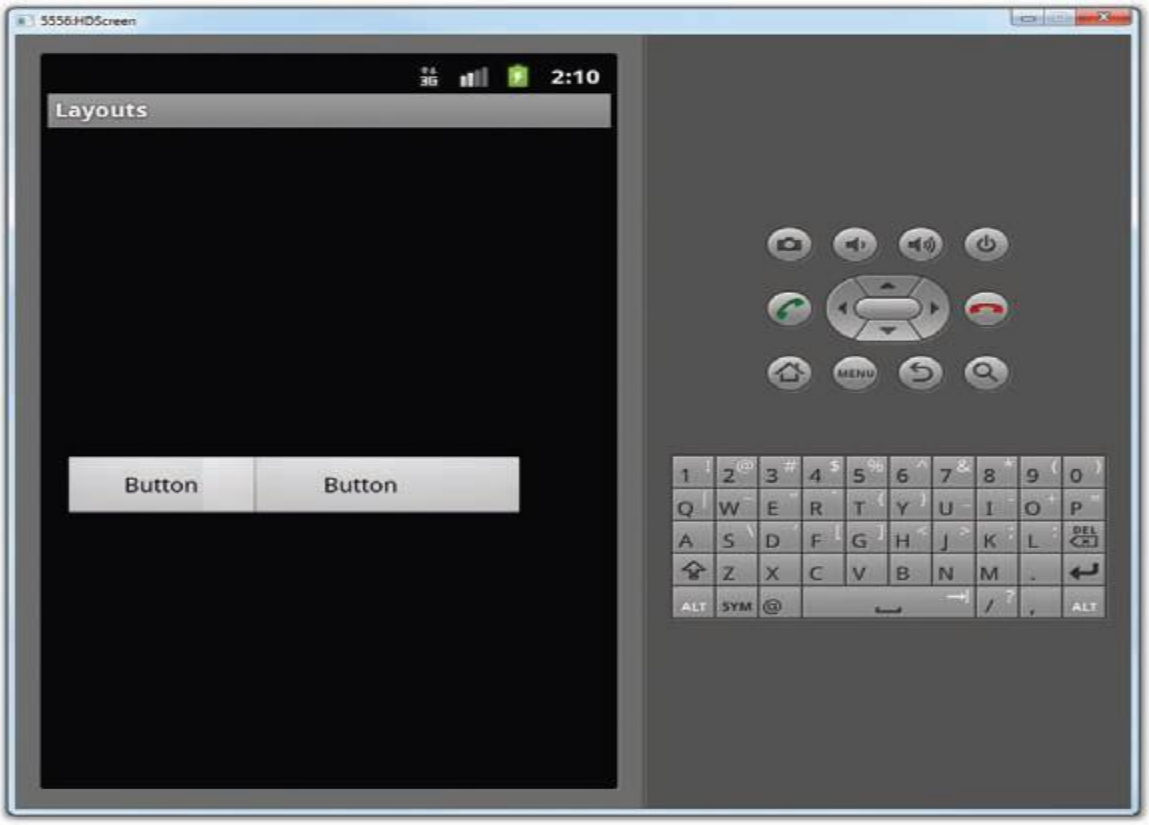

Figure 3-6