Chapter 10

Working with Containers

A container pours a collection of widgets (and possibly child containers) into a specific structure of your choosing. If you want a form with labels on the left and fields on the right, you need a container. If you want OK and Cancel buttons to be beneath the rest of the form, next to one another, and flush to the right side of the screen, you need a container. Just from a pure XML perspective, if you have multiple widgets (beyond RadioButton widgets in a RadioGroup), you need a container just to have a root element in which to place the widgets.

Most GUI toolkits have some notion of layout management, frequently organized into containers. In Java/Swing, for example, you have layout managers like BoxLayout and containers that use them (e.g., Box). Some toolkits, such as XUL and Flex, stick strictly to the box model, figuring that any desired layout can be achieved through the right combination of nested boxes. Android, through LinearLayout, also offers a box model, but in addition supports a range of containers that provide different layout rules.

In this chapter, we will look at four commonly used containers, LinearLayout (the box model), RelativeLayout (a rule-based model), and TableLayout (the grid model), along with the all-new GridLayout (the infinite fine-line model) released with Ice Cream Sandwich (ICS). We’ll also look at ScrollView, a container designed to assist with implementing scrolling containers.

Thinking Linearly

As just noted, LinearLayout is a box model—widgets or child containers are lined up in a column or row, one after the next. This works similarly to FlowLayout in Java/Swing, vbox and hbox in Flex and XUL, and so forth.

Flex and XUL use the box as their primary unit of layout. If you want, you can use LinearLayout in much the same way, eschewing some of the other containers. Getting the visual representation you want is mostly a matter of identifying where boxes should nest and which properties those boxes should have, such as their alignment relative to other boxes.
LinearLayout Concepts and Properties

To configure a LinearLayout, you have five main areas of control besides the container’s contents: the orientation, the fill model, the weight, the gravity, and the padding.

**Orientation**

Orientation indicates whether the LinearLayout represents a row or a column. Just add the `android:orientation` property to your `LinearLayout` element in your XML layout, and set the value to be `horizontal` for a row or `vertical` for a column.

The orientation can be modified at runtime by invoking `setOrientation()` on the `LinearLayout`, supplying it either `HORIZONTAL` or `VERTICAL`.

**Fill Model**

Imagine a row of widgets, such as a pair of radio buttons. These widgets have a “natural” size based on their text. Their combined size probably does not exactly match the width of the Android device’s screen—particularly since screens come in various sizes. We then have the issue of what to do with the remaining space.

All widgets inside a LinearLayout must supply `android:layout_width` and `android:layout_height` properties to help address this issue. These properties’ values have three flavors:

- You can provide a specific dimension, such as `125dp`, to indicate the widget should take up exactly a certain size.
- You can provide `wrap_content`, which means the widget should fill up its natural space, unless that is too big, in which case Android can use word-wrap as needed to make it fit.
- You can provide `fill_parent`, which means the widget should fill up all available space in its enclosing container, after all other widgets are taken care of.

The latter two flavors are the most common, as they are independent of screen size, allowing Android to adjust your view to fit the available space.

**NOTE:** In API level 8 (Android 2.2), `fill_parent` was renamed to `match_parent`, for unknown reasons. You can still use `fill_parent`, as it will be supported for the foreseeable future. However, at such point in time as you are supporting only API level 8 or higher (e.g., `android:minSdkVersion="8"` in your manifest), you should probably switch over to `match_parent`. 