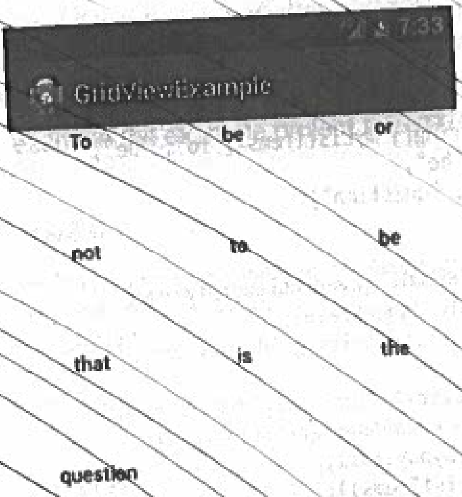
Spinner

~~By choosing large verticalSpacing and columnWidth values in our layout (both 75dip), I am attempting to create large box-like cells in the resulting application. In fact, such large cells are almost like implied buttons. This effect is useful, and you can even go so far as to have widgets like Buttons or ImageViews as the content of your cells (remember, you are in complete control of the layout the adapter uses). Figure 7-3 shows our grid as it first appears.~~

~~Remember that your emulator or device might have a different screen size, and so our android:numColumns="auto_fit" value can result in a layout of differing numbers of rows/columns than the one you see in Figure 7-3.~~



To be or
not to be
that is the
question

Figure 7-3. GridView with data from an ArrayAdapter

Spinners.

Taking Options for a Spin

There are times when you want to provide your users with a full list of items from which to choose, but for other design reasons or constraints, you simply don't have the space to show them the entire list in one go. Other widget toolkits address this with the notion of a drop-down or pick list. In Android, the same approach to saving space is achieved with the spinner.

Even without code of your own, you can see how a spinner works by simply playing with the time and date settings of any Android device. In these cases, the source data is taken from the system clock. When designing your own spinners, the general approach is the same as the one you took with ListView or GridView: create an adapter to provide the data you want displayed, pick an appropriate view layout for your spinner "rows," and hook up a listener object with setOnItemSelectedListener() to carry out your desired logic when a user makes their choice.

Unlike other selection widgets, a spinner has two visual forms: the collapsed version and the dropped-down version that appears while selection is in progress. If you want to also customize the look and feel of your spinner in its dropped-down state, you still configure the adapter (just as you do for the regular state of all selection widgets) and not the Spinner widget itself. You do this with a call to the setDropDownViewResource() method, where you provide the necessary resource ID of your desired view for the dropped-down state.

Listing 7-10 shows our ongoing Hamlet example converted to use a spinner.

Listing 7-10. Layout XML for the Spinner Widget

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SpinnerExample" >

    <TextView
        android:id="@+id/mySelection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Spinner android:id="@+id/spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="true" />

</RelativeLayout>
```

XML
Widget = Spinner

The attribute android:drawSelectorOnTop controls whether the arrow that provides the hint that this is a Spinner widget is drawn on the side of the spinner UI.

You can now pour in the Java code that should look mostly familiar to you by now, with the necessary substitutions to populate and use the spinner, as shown in Listing 7-11:

Listing 7-11. Java Code to Support the Spinner Widget

```
package com.artifexdigital.android.spinnerexample;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
import android.app.Activity;
```

```
public class SpinnerExample extends Activity
    implements AdapterView.OnItemSelectedListener {
```

global

```
    private TextView mySelection;
    private static final String[] myListItems={"To", "be",
        "or", "not", "to", "be",
        "that", "is", "the", "question"};
```

onCreate

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_spinner_example);

        mySelection=(TextView)findViewById(R.id.mySelection);

        Spinner spin=(Spinner)findViewById(R.id.spinner);
        spin.setOnItemSelectedListener(this);

        ArrayAdapter<String> myAdapter=new ArrayAdapter<String>(this,
                                   android.R.layout.simple_spinner_item,
                                   myListItems);

        myAdapter.setDropDownViewResource(
            android.R.layout.simple_spinner_dropdown_item);
        spin.setAdapter(myAdapter);
    }
```

2 new methods

```
    public void onItemSelected(AdapterView<?> parent,
        View v, int position, long id) {
        mySelection.setText(myListItems[position]);
    }

    public void onNothingSelected(AdapterView<?> parent) {
        mySelection.setText("");
    }
}
```

In the Java implementation, when you use `spin.setOnItemSelectedListener(this)`, the activity itself is designated as the selection listener. You can do this because the activity implements the `OnItemSelectedListener` interface. As I described earlier, it is possible to have a custom `View` for both the collapsed and dropped-down states of a spinner, and the call to `aa.setDropDownViewResource()` achieves this. I have used `android.R.layout.simple_spinner_item` as the `View` for each row on the spinner—this is another of the defaults

ener(this), the
ause the activity
r, it is possible to
a spinner, and
roid.R.layout.
other of the defaults

shipped with the SDK. OnItemSelectedListener() updates the other label widget with the chosen selection just as we did with the ListView and GridView examples.

Figures 7-4 and 7-5 show the spinner in action.

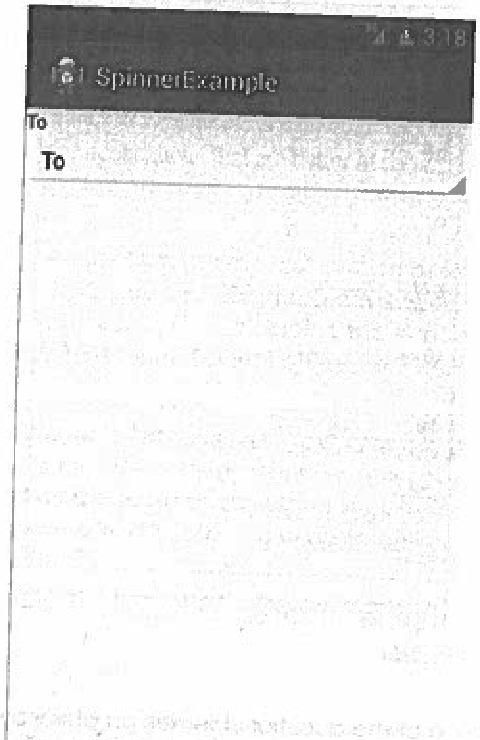


Figure 7-4. Spinner showing initial state/collapsed state