```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="rectangle">
  <gradient
    android:startColor="#44FF0000"
    android:endColor="#FFFF0000"
    android:angle="270"
  />
  <padding
    android:top="2px"
    android:bottom="2px"
  />
  <corners android:radius="6px" />
</shape>
```

A gradient is applied to the more general-purpose `<shape>` element, in this case, a rectangle. The gradient is defined as having a start and end color – in this case, the gradient is an increasing amount of red, with only the alpha channel varying to control how much the background blends in. The color is applied in a direction determined by the number of degrees specified by the `android:angle` attribute, with `270` representing "down" (start color at the top, end color at the bottom).

As with any other XML-defined shape, you can control various aspects of the way the shape is drawn. In this case, we put some padding around the drawable and round off the corners of the rectangle.

To use this `Drawable` in Java code, you can reference it as `R.drawable.active_row`. One possible use of a gradient is in custom `ListView` row selection, as shown in `Drawable/GradientDemo`:

```
package com.commonsware.android.drawable;

import android.app.ListActivity;
import android.content.Context;
import android.os.Bundle;
import android.content.res.ColorStateList;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class GradientDemo extends ListActivity {
  private static ColorStateList allWhite=ColorStateList.valueOf(0xFFFFFFFF);
  private static String[] items={"lorem", "ipsum", "dolor",
```

```
                                "sit", "amet", "consectetuer",
                                "adipiscing", "elit", "morbi",
                                "vel", "ligula", "vitae",
                                "arcu", "aliquet", "mollis",
                                "etiam", "vel", "erat",
                                "placerat", "ante",
                                "porttitor", "sodales",
                                "pellentesque", "augue",
                                "purus"};

@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  setContentView(R.layout.main);
  setListAdapter(new GradientAdapter(this));
  getListView().setOnItemSelectedListener(listener);
}

class GradientAdapter extends ArrayAdapter {
  GradientAdapter(Context ctxt) {
    super(ctxt, R.layout.row, items);
  }

  @Override
  public View getView(int position, View convertView,
                      ViewGroup parent) {
    GradientWrapper wrapper=null;

    if (convertView==null) {
      convertView=getLayoutInflater().inflate(R.layout.row,
                                              null);
      wrapper=new GradientWrapper(convertView);
      convertView.setTag(wrapper);
    }
    else {
      wrapper=(GradientWrapper)convertView.getTag();
    }

    wrapper.getLabel().setText(items[position]);

    return(convertView);
  }
}

class GradientWrapper {
  View row=null;
  TextView label=null;

  GradientWrapper(View row) {
    this.row=row;
  }

  TextView getLabel() {
    if (label==null) {
```

```
        label=(TextView)row.findViewById(R.id.label);
      }

      return(label);
    }
  }

  AdapterView.OnItemSelectedListener listener=new
AdapterView.OnItemSelectedListener() {
    View lastRow=null;

    public void onItemSelected(AdapterView<?> parent,
                               View view, int position,
                               long id) {
      if (lastRow!=null) {
        lastRow.setBackgroundColor(0x00000000);
      }

      view.setBackgroundResource(R.drawable.active_row);
      lastRow=view;
    }

    public void onNothingSelected(AdapterView<?> parent) {
      if (lastRow!=null) {
        lastRow.setBackgroundColor(0x00000000);
        lastRow=null;
      }
    }
  };
}
```

In an earlier chapter, we showed how you can get control and customize how a selected row appears in a `ListView`. This time, we apply the gradient rounded rectangle as the background of the row. We could have accomplished this via appropriate choices for `android:listSelector` and `android:drawSelectorOnTop` as well.

The result is a selection bar implementing the gradient:

Figure 20. The GradientDemo sample application

Note that because the list background is black, the red is mixed with black on the top end of the gradient. If the list background were white, the top end of the gradient would be red mixed with white, as determined by the alpha channel specified on the gradient's top color.

A Stitch In Time Saves Nine

As you read through the Android documentation, you no doubt ran into references to "nine-patch" or "9-patch" and wondered what Android had to do with quilting. Rest assured, you will not need to take up needlework to be an effective Android developer.

If, however, you are looking to create backgrounds for resizable widgets, like a `Button`, you will probably need to work with nine-patch images.

As the Android documentation states, a nine-patch is "a PNG image in which you define stretchable sections that Android will resize to fit the object at display time to accommodate variable sized sections, such as text strings". By using a specially-created PNG file, Android can avoid trying to

use vector-based formats (e.g., SVG) and their associated overhead when trying to create a background at runtime. Yet, at the same time, Android can still resize the background to handle whatever you want to put inside of it, such as the text of a `Button`.

In this section, we will cover some of the basics of nine-patch graphics, including how to customize and apply them to your own Android layouts.

The Name and the Border

Nine-patch graphics are PNG files whose names end in `.9.png`. This means they can be edited using normal graphics tools, but Android knows to apply nine-patch rules to their use.

What makes a nine-patch graphic different than an ordinary PNG is a one-pixel-wide border surrounding the image. When drawn, Android will remove that border, showing only the stretched rendition of what lies inside the border. The border is used as a control channel, providing instructions to Android for how to deal with stretching the image to fit its contents.

Padding and the Box

Along the right and bottom sides, you can draw one-pixel-wide black lines to indicate the "padding box". Android will stretch the image such that the contents of the widget will fit inside that padding box.

For example, suppose we are using a nine-patch as the background of a `Button`. When you set the text to appear in the button (e.g., "Hello, world!"), Android will compute the size of that text, in terms of width and height in pixels. Then, it will stretch the nine-patch image such that the text will reside inside the padding box. What lies outside the padding box forms the border of the button, typically a rounded rectangle of some form.

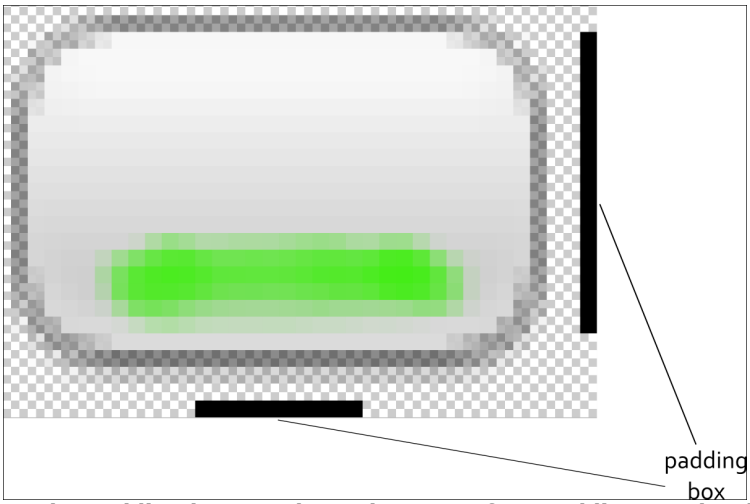

Figure 21. The padding box, as shown by a set of control lines to the right and bottom of the stretchable image

Stretch Zones

To tell Android where on the image to actually do the stretching, draw one-pixel-wide black lines on the top and left sides of the image. Android will scale the graphic only in those areas – areas outside the stretch zones are not stretched.

Perhaps the most common pattern is the center-stretch, where the middle portions of the image on both axes are considered stretchable, but the edges are not:

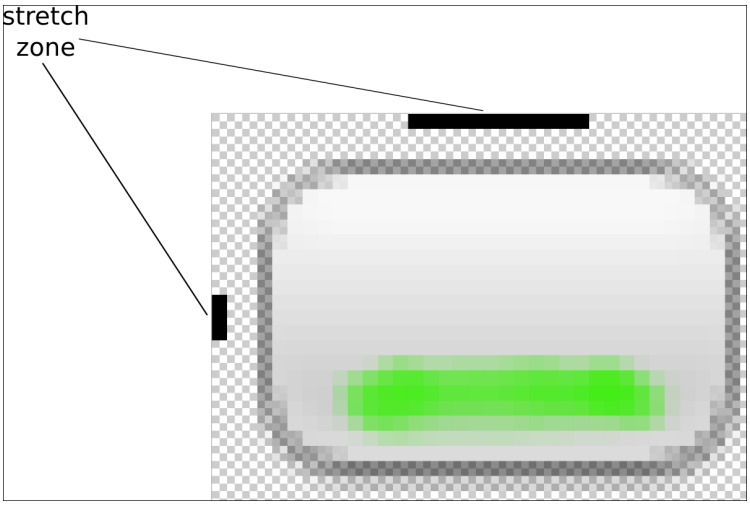

Figure 22. The stretch zones, as shown by a set of control lines to the right and bottom of the stretchable image

Here, the stretch zones will be stretched just enough for the contents to fit in the padding box. The edges of the graphic are left unstretched.

Some additional rules to bear in mind:

- If you have multiple discrete stretch zones along an axis (e.g., two zones separated by whitespace), Android will stretch both of them but keep them in their current proportions. So, if the first zone is twice as wide as the second zone in the original graphic, the first zone will be twice as wide as the second zone in the stretched graphic.
- If you leave out the control lines for the padding box, it is assumed that the padding box and the stretch zones are one and the same.

Tooling

To experiment with nine-patch images, you may wish to use the `draw9patch` program, found in the `tools/` directory of your SDK installation:

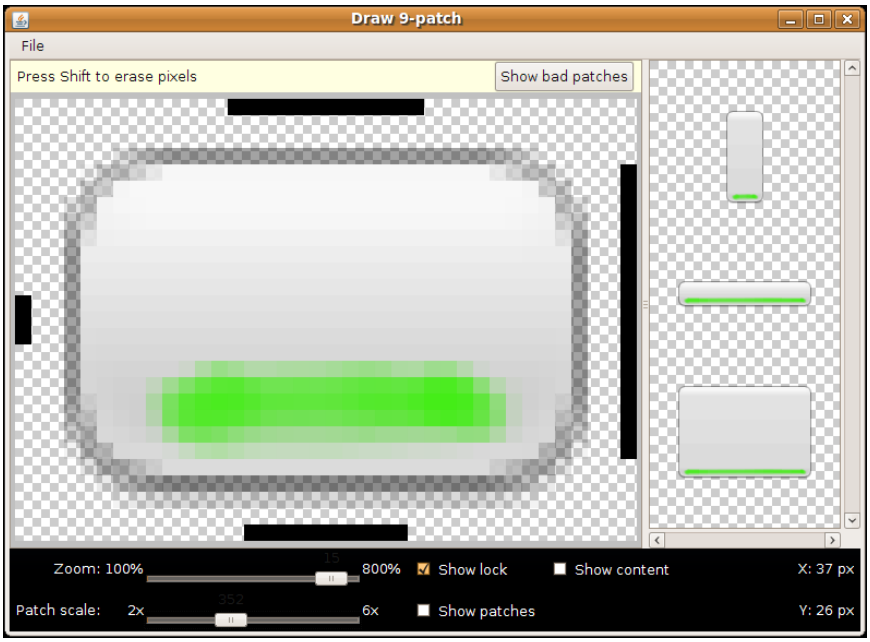

Figure 23. The draw9patch tool

While a regular graphics editor would allow you to draw any color on any pixel, `draw9patch` limits you to drawing or erasing pixels in the control area. If you attempt to draw inside the main image area itself, you will be blocked:

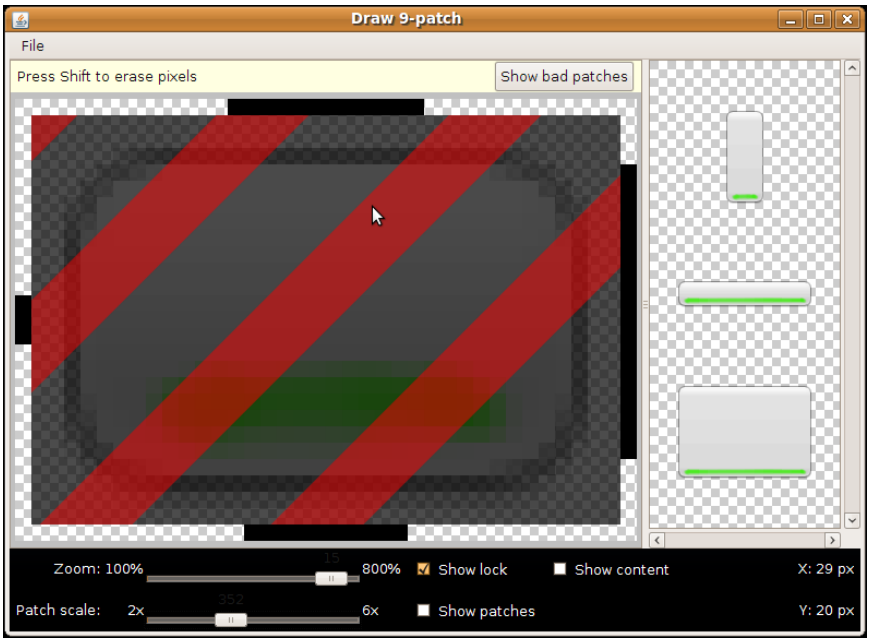

Figure 24. The draw9patch tool, showing blocked areas

On the right, you will see samples of the image in various stretched sizes, so you can see the impact as you change the stretchable zones and padding box.

While this is convenient for working with the nine-patch nature of the image, you will still need some other graphics editor to create or modify the body of the image itself. For example, the image shown above, from the `Drawable/NinePatch` project, is a modified version of a nine-patch graphic from the SDK's `ApiDemos`, where the GIMP was used to add the neon green stripe across the bottom portion of the image.

Using Nine-Patch Images

Nine-patch images are most commonly used as backgrounds, as illustrated by the following layout:

```
<?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:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="1"
  >
    <TableRow
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
    >
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:text="Horizontal:"
      />
      <SeekBar android:id="@+id/horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
      />
    </TableRow>
    <TableRow
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
    >
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:text="Vertical:"
      />
      <SeekBar android:id="@+id/vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
      />
    </TableRow>
  </TableLayout>
  <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button android:id="@+id/resize"
      android:layout_width="48px"
      android:layout_height="48px"
      android:text="Hi!"
      android:background="@drawable/button"
    />
  </LinearLayout>
</LinearLayout>
```

Here, we have two SeekBar widgets, labeled for the horizontal and vertical axes, plus a Button set up with our nine-patch graphic as its background (android:background = "@drawable/button").

The NinePatchDemo activity then uses the two SeekBar widgets to let the user control how large the button should be drawn on-screen, starting from an initial size of 48px square:

```
package com.commonsware.android.drawable;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.SeekBar;

public class NinePatchDemo extends Activity {
  SeekBar horizontal=null;
  SeekBar vertical=null;
  View thingToResize=null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    thingToResize=findViewById(R.id.resize);

    horizontal=(SeekBar)findViewById(R.id.horizontal);
    vertical=(SeekBar)findViewById(R.id.vertical);

    horizontal.setMax(272);  // 320 less 48 starting size
    vertical.setMax(272);    // keep it square @ max

    horizontal.setOnSeekBarChangeListener(h);
    vertical.setOnSeekBarChangeListener(v);
  }

  SeekBar.OnSeekBarChangeListener h=new SeekBar.OnSeekBarChangeListener() {
    public void onProgressChanged(SeekBar seekBar,
                                  int progress,
                                  boolean fromTouch) {
      ViewGroup.LayoutParams old=thingToResize.getLayoutParams();
      ViewGroup.LayoutParams current=new LinearLayout.LayoutParams(48+progress,
                                                                   old.height);

      thingToResize.setLayoutParams(current);
    }

    public void onStartTrackingTouch(SeekBar seekBar) {
```

```
      // unused
    }

    public void onStopTrackingTouch(SeekBar seekBar) {
      // unused
    }
  };

  SeekBar.OnSeekBarChangeListener v=new SeekBar.OnSeekBarChangeListener() {
    public void onProgressChanged(SeekBar seekBar,
                                  int progress,
                                  boolean fromTouch) {
      ViewGroup.LayoutParams old=thingToResize.getLayoutParams();
      ViewGroup.LayoutParams current=new LinearLayout.LayoutParams(old.width,
                                                                   48+progress);

      thingToResize.setLayoutParams(current);
    }

    public void onStartTrackingTouch(SeekBar seekBar) {
      // unused
    }

    public void onStopTrackingTouch(SeekBar seekBar) {
      // unused
    }
  };
}
```

The result is an application that can be used much like the right pane of `draw9patch`, to see how the nine-patch graphic looks on an actual device or emulator in various sizes:

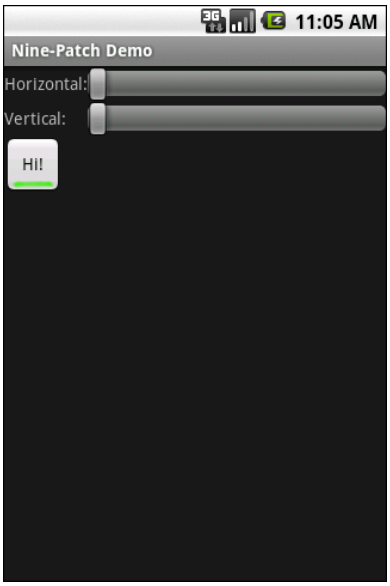

Figure 25. The NinePatch sample project, in its initial state

Figure 26. The NinePatch sample project, after making it bigger horizontally

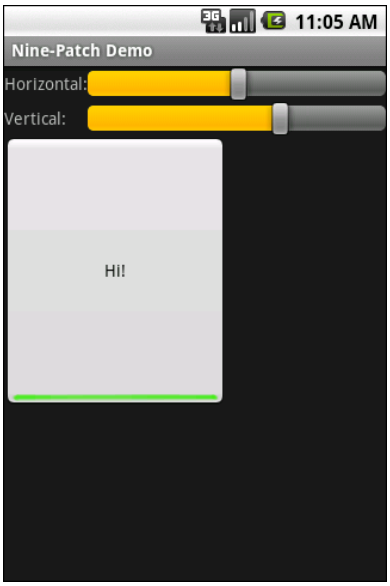

Figure 27. The NinePatch sample application, after making it bigger in both dimensions