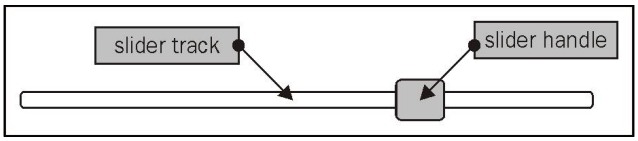
6
The Slider Widget

The slider component allows us to implement an engaging and easy-to-use widget that our visitors should find attractive and intuitive to use. Its basic function is simple. The slider background represents a series of values that are selected by dragging the thumb along the background.

In this chapter, we will cover the following topics:

- The default slider implementation
- Custom styling for sliders
- Changing configuration options
- Creating a vertical slider
- Setting minimum, maximum, and default values
- Enabling multiple handles and ranges
- The slider's built-in event callbacks
- Slider methods

Before we roll up our sleeves and begin creating a slider, let's look at the different elements that it is made from. The following screenshot shows a typical slider widget:



It's a simple widget, as you can see, comprised of just two main elements—the slider handle, also called the thumb, and the slider background, also called the track.

Implementing a slider

Creating the default, basic slider takes no more code than any of the other widgets that we have looked at so far. The underlying HTML markup required is also minimal. Let's create a basic one now. In a new page in your text editor, add the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet"
      href="css/smoothness/jquery-ui-1.8.9.custom.css">
    <title>Slider
    </title>
  </head>
  <body>
    <div id="mySlider">
    </div>
    <script src="development-bundle/jquery-1.4.4.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.core.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.widget.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.mouse.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.slider.js">
    </script>
    <script>
      (function($){
        $("#mySlider").slider();
      })(jQuery);
    </script>
  </body>
</html>
```

Save this file as `slider1.html` and view it in your browser. On the page is a simple container element; this will be transformed by the widget into the slider track. In the `<script>` at the end of the `<body>`, we select this element and call the `slider` method on it. The `<a>` element that is used for the slider handle will be automatically created by the widget.

When we run this page in a browser, we should see something similar to the previous screenshot. We've used several library resources for the default implementation, including the following files:

- `jquery-ui-x.x.x.custom.css`
- `jquery-x.x.x.js`
- `jquery.ui.core.js`
- `jquery.ui.widget.js`
- `jquery.ui.mouse.js`
- `jquery.ui.slider.js`

The default behavior of a basic slider is simple but effective. The thumb can be moved horizontally along any pixel of the track on the `x`-axis by dragging the thumb with the mouse pointer, or using the *left/down* or *right/up* arrow keys of the keyboard. Clicking anywhere on the track with the left or right mouse button will instantly move the handle to that position.

Custom styling

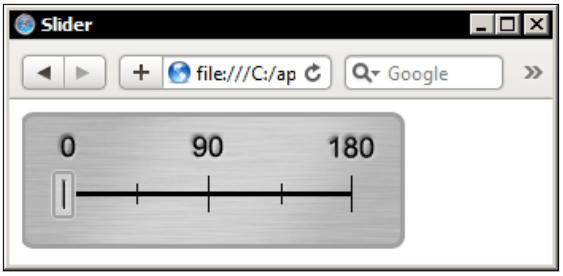
Because of its simplicity, the slider widget is extremely easy to create a custom theme for. Using ThemeRoller is the recommended method of theming, but to completely change the look and feel of the widget, we can easily create our own theme file. In your text editor create the following style sheet:

```
. background-div {
  height:50px; width:217px; padding:36px 0 0 24px;
  background:url(../img/slider_outerbg.gif) no-repeat;
}
#mySlider {
  width:184px; height:23px; border:none; position:relative;
  left:4px; top:4px;
  background:url(../img/slider_bg.gif) no-repeat;
}
#mySlider .ui-slider-handle {
  width:14px; height:30px; top:-4px;
  background:url(../img/slider_handle.gif) no-repeat;
}
```

Save this file as `sliderTheme.css` in the `css` directory. In `slider1.html`, add a link to the style sheet in the `<head>` of the page (after the jQuery UI style sheet), and wrap the underlying slider element in a new container:

```
<div class="background-div">
  <div id="mySlider">
  </div>
</div>
```

Resave the file as `slider2.html`. With a minimum of CSS and a few images (these can be found in the code download), we can easily but considerably modify the widget's appearance, as shown in the following screenshot:



Of course, this example is completely arbitrary, and was intended purely to show how to override the default theme.

Configurable options

Additional functionality, such as vertical sliders, multiple handles, and stepping can also be configured using an object literal, passed into the widget method when the slider is initialized. The options that can be used in conjunction with the slider widget are listed in the following table:

Option	Default value	Used to…
`animate`	`false`	Enable a smooth animation of the slider handle when the track is clicked.
`disabled`	`false`	Disable the widget when it is initialized.
`max`	`100`	Set the maximum value of the slider.
`min`	`0`	Set the minimum value of the slider.
`orientation`	`auto`	Set the axis along which the slider thumb is moved. This can accept the strings `vertical` or `horizontal`.

Option	Default value	Used to…
`range`	`false`	Create a style-able range element between them.
`step`	`1`	Set the distance of the step the handle will take along the track. The `max` value must be equally divisible by the supplied number.
`value`	`0`	Set the value of the slider thumb when the widget is initialized.
`values`	`null`	Accept an array of values. Each supplied integer will become the value of a slider handle.

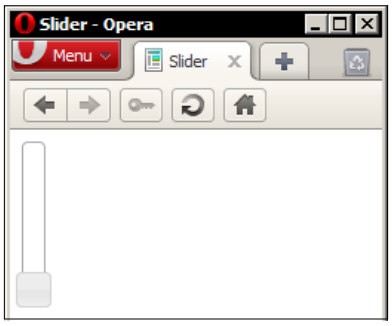
Creating a vertical slider

To make a vertical slider, all we need to do is set the `orientation` option to `vertical`; the widget will do the rest for us.

In `slider1.html`, change the final `<script>` element so that it appears as follows:

```
<script>
  (function($){
    var sliderOpts = {
      orientation: "vertical"
    };
    $("#mySlider").slider(sliderOpts);
  })(jQuery);
</script>
```

Save the new file as `slider3.html`. We just need to set this single option to put the slider into `vertical` mode. When we launch the page, we see that the slider operates exactly as it did before, except that it now moves along the `y`-axis, as in the following screenshot:



The widget defaults to `100 px` in height, if we don't provide our own CSS `height` rule for the slider track.

Minimum and maximum values

By default, the minimum value of the slider is `0` and the maximum value is `100`, but we can change these values easily using the `min` and `max` options. Change the configuration object in `slider3.html` to the following:

```
var sliderOpts = {
  min: -50,
  max: 50
};
```

Save this as `slider4.html`. We simply specify the integers that we'd like to set as the starting and end values. Because the `value` option is set to `0` by default, when we run this file, the slider thumb will start in the middle of the track, half way between `-50` and `50`.

When the slider handle, in this example, is at the minimum value, the `value` method (see the *methods* section) will return `-50`, as we would expect.

Slider steps

The `step` option refers to the number and position of steps along the track that the slider's handle jumps, when moving from the minimum to the maximum positions on the track. The best way to understand how this option works is to see it in action, so change the configuration object in `slider4.html` to the following:

```
var sliderOpts = {
  step: 25
};
```

Save this version as `slider5.html`. We set the `step` option to `25` in this example. We haven't set the `min` or `max` options, so they will take the default values of `0` and `100` respectively. Hence, by setting `step` to `25`, we're saying that each step along the track should be a quarter of the track's length, because `100` (the maximum) divided by `25` (the step value) is `4`. The thumb will therefore take four steps along the track, from beginning to end.

The `max` value of the slider should be equally divisible by whatever value we set as the `step` option, other than that, we're free-to-use whatever value we wish. This is a useful option for confining the value selected by visitors, to one of a set of predefined values.

If we were to set the value of the `step` option, in this example, to `27` instead of `25`, the slider would still work, but the points along the track that the handle stepped to, would not be equal.

Slider animation

The slider widget comes with a built-in animation that moves the slider handle smoothly to a new position, whenever the slider track is clicked. This animation is disabled by default, but we can easily enable it by setting the `animate` option to `true`. Change the configuration object in `slider5.html`, so that it is as follows:

```
var sliderOpts = {
  animate: true
}
```

Save this version as `slider6.html`. The difference this option makes to the overall effect of the widget is extraordinary. Instead of the slider handle just moving instantly to a new position when the track is clicked, it smoothly slides there.

If the `step` option is configured to a value other than `1`, and the `animate` option is enabled, the thumb will slide to the nearest step mark on the track. This may mean that the slider thumb moves past the point that was clicked.

Setting the slider's value

The `value` option, when set to `true` in a configuration object, determines the starting value for the slider thumb. Depending on what we want the slider to represent, the starting value of the handle may not be `0`. If we wanted to start at half-way across the track instead-of at the beginning, we could use the following configuration object:

```
var sliderOpts = {
  value: 50
}
```

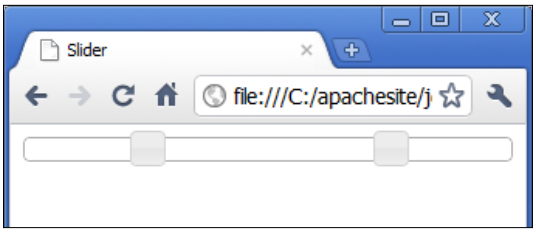
Save this file as `slider7.html`. When the file is loaded in a browser, we see that the handle starts halfway along the track instead of at the beginning, exactly as it did when we set the `min` and `max` options earlier on. We can also set this option after initialization, to programmatically set a new value.

Using multiple handles

I mentioned earlier that a slider may have multiple handles; additional handles can be added using the `values` option. It accepts an array, where each item in the array is a starting point for a handle. We can specify as many items as we wish, up to the `max` value (taking `step` into account):

```
var sliderOpts = {
  values: [25, 75]
};
```

Save this variation as `slider8.html`. This is all we need to do; we don't need to supply any additional underlying markup. The widget has created both new handles for us, and as you'll see, they both function exactly as a standard single handle does. The following screenshot shows our dual-handled slider:



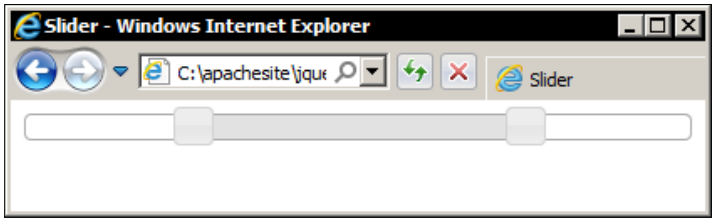
When a slider has two or more handles, each handle may move past the other handle(s) without issue.

The range element

When working with multiple handles, we can set the `range` option to `true`. This adds a styled range element between two handles. In `slider8.html`, change the configuration object to this:

```
var sliderOpts = {
  values: [25, 75],
  range: true
};
```

Save this page as `slider9.html`. When the page loads, we should see that a styled `<div>` element now connects our two handles, as in the following screenshot:



A maximum of two handles can be used in conjunction with the `range` option, but we can also enable it with a single handle as well; change the configuration object in the previous example to the following:

```
var sliderOpts = {
  range: "min"
};
```

Save this as `slider10.html`. Along with the Boolean value `true`, we can also supply one of the string values `"min"` or `"max"`, but only when a single handle is in use.

In this example, we set it to `"min"`, so when we move the slider handle along the track, the range element will stretch from the start of the track to the slider handle. If we set the option to `"max"`, the range will stretch from the handle to the end of the track.

When using two handles and a range, the two handles may not cross each other on the track.

Using slider's event API

In addition to the options we saw earlier, there are another four options used to define functions that are executed at different times during a slider interaction. Any callback functions that we use, are automatically passed by the standard event object and an object representing the slider. The following table lists the event options we can use:

Event	Fired when…
`change`	The slider's handle stops moving, and its value has changed.
`slide`	The slider's handle moves.
`start`	The slider's handle starts moving.
`stop`	The slider's handle stops moving.

Hooking into these built-in callback functions is extremely easy. Let's put a basic example together to see. Change the configuration object in `slider10.html`, so that it appears as follows:

```
var sliderOpts = {
  start: function() {
    $("#tip").fadeOut(function() {
      $(this).remove();
    });
  },
  change: function(e, ui) {
    $("<div></div>", {
      "class": "ui-widget-header ui-corner-all",
      id: "tip",
      text: ui.value + "%",
      css: {
        left: e.pageX - 35
```

```
        }
      }).appendTo("#mySlider");
    }
};
```

Save this as `slider11.html`. We use two of the callback options in this example—`start` and `change`. In the `start` function, we select the tool tip element if it exists, and fade it out with jQuery's `fadeOut()` method. Once hidden from view, it is removed from the page.

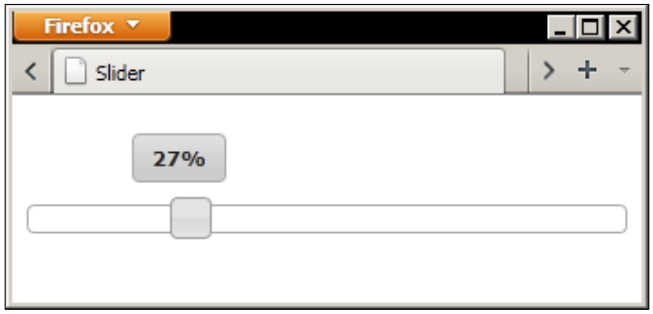
The `change` function will be executed each time the value of the slider handle changes; when the function is called, we create the tool tip and append it to the slider. We position it so that it appears above the center of the slider handle and give it some of the framework classnames in order to style it with the theme in use.

In several places, we use the second object passed to the callback function, the prepared `ui` object that contains useful information from the slider. In this example, we use the `value` option of the object to obtain the new value of the slider handle.

We also need a very small custom style sheet for this example. In a new page in your text editor, add the following code:

```
#mySlider { margin:60px auto 0; }
#tip {
  position:absolute; display:inline; padding:5px 0;
  width:50px; text-align:center; font:bold 11px Verdana;
}
```

Save this file as `sliderTheme2.css` in the `css` folder, and link to it from the `<head>` of `slider11.html`. When displayed, our tool tip should appear as shown in the following screenshot:



When all of the event options are used together, the events will be fired in the following order:

- `start`
- `slide`
- `stop`
- `change`

The `slide` callback can be quite an intensive event as it is fired on every mouse-move while the handle is selected, but it can also be used to prevent a slide in certain situations by returning `false` from the callback function. When using the `stop` and `change` callbacks together, the `change` callback may override the `stop` callback.

As with all library components, each of these events can also be used with jQuery's `bind()` method by prefixing the word `slider` to the event name, for example, `sliderstart`.

Slider methods

The slider is intuitive and easy-to-use, and like the other components in the library, it comes with a range of methods that are used to programmatically control the widget after it has been initialized. The methods specific to the slider are shown in the following table:

Method	Used to…
`value`	Set a single slider handle to a new value. This will move the handle to the new position on the track automatically. This method accepts a single argument which is an integer representing the new value.
`values`	Set the specified handle to move to a new value when multiple handles are in use. This method is exactly the same as the `value` method, except that it takes two arguments—the index number of the handle followed by the new value.

The methods `destroy`, `disable`, `enable`, `option`, and `widget` are common to all components, and work in the same way with slider that we would expect them to.

The `value` and `values` methods are exclusive to the slider, and are used to get or set the value of single or multiple handles. Of course, we can also do this using the `option` method, so these two methods are merely shortcuts to cater for common implementational requirements. Let's take a look at them in action. First of all, let's see how the `value` method can be used.

In `slider11.html`, remove the `<link>` to `sliderTheme2.css` and add a new `<button>` element to the page, directly after the slider container:

```
<button type="button" id="setMax">Set to max value</button>
```

Now, change the final `<script>` element so that it is as follows:

```
<script>
  (function($){
    $("#mySlider").slider();
    $("#setMax").click(function() {
      var maxVal = $("#mySlider").slider("option", "max");
      $("#mySlider").slider("value", maxVal);
    });
  })(jQuery);
</script>
```

Save this file as `slider12.html`. We add a click handler for our new `<button>`; whenever it is clicked, this method will first determine what the maximum value for the slider is, by setting a variable to the result of the `option` method, specifying `max` as the option we'd like to get. We don't need a configuration object in this example.

Once we have the maximum value, we then call the `value` method, passing in the variable that holds the maximum value as the second argument; our variable will be used as the new value. Whenever the button is clicked, the slider handle will instantly move to the end of the track.

Working with multiple handles is just as easy, but involves a slightly different approach. Remove the `setMax` button in `slider12.html`, and add these two buttons directly after the slider element:

```
<button type="button" class="preset" id="low">Preset 1 (low)
</button>
<button type="button" class="preset" id="high">Preset 2 (high)
</button>
```

Now change the final `<script>` element at the end of the `<body>` to the following:

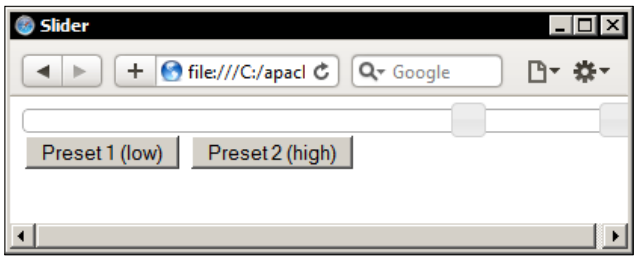
```
<script>
  $(function(){

    var sliderOpts = {
      values: [25, 75]
    };

    $("#mySlider").slider(sliderOpts);
    $(".preset").click(function() {
```

```
        if (this.id === "low") {
          $("#mySlider").slider("values", 0, 0);
          $("#mySlider").slider("values", 1, 25);
        } else {
          $("#mySlider").slider("values", 0, 75);
          $("#mySlider").slider("values", 1, 100);
        }
      });
    });
  </script>
```

Save this file as `slider13.html`. To trigger multiple handles, we specify the values of two handles in our configuration object. When either of the two `<button>` elements on the page are clicked, we work out which button was clicked and then set the handles to either low values or high values, depending on which button was clicked.

The `values` method takes two arguments. The first argument is the index number of the handle we'd like to change, and the second argument is the value that we'd like the handle to be set to. Notice that we have to set each handle individually and that we can't chain the two methods together. This is because the method returns the new value and not a jQuery object (this is fixed in an upcoming version of the library).

The following screenshot shows how the page should appear after the second button is clicked:



Practical uses

An HTML5 element that may lend itself particularly well to implementations of the slider widget, is the `<audio>` element. This element will automatically add controls that enable the visitor to play, pause, and adjust the volume of the media being played.

The default controls, however, at this point anyway, do not appear to be style-able, so if we wish to change their appearance, we need to create our own controls. The slider widget of course, makes an excellent substitution for the default volume control.

Create the following new page in your text editor:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet"
      href="css/smoothness/jquery-ui-1.8.9.custom.css">
    <meta charset="utf-8">
    <title>Slider
    </title>
  </head>
  <body>
    <audio id="audio"
      src="http://upload.wikimedia.org/wikipedia/
      en/7/77/Jamiroquai_-_Snooze_You_Lose.ogg">
      Your browser does not support the <code>audio</code>
        element.
    </audio>
    <div id="volume">
    </div>
    <script src="development-bundle/jquery-1.3.2.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.core.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.widget.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.mouse.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.slider.js">
    </script>
    <script>
      (function($){
        var audio = $("audio")[0],
        sliderOpts = {
          value: 5,
          min: 0,
          max: 10,
          orientation: "vertical",
          change: function() {
            var vol = $(this).slider("value") / 10;
```

```
            audio.volume = vol;
          }
        };
        audio.volume = 0.5;
        audio.play();
        $("#volume").slider(sliderOpts);
      })(jQuery);
    </script>
  </body>
</html>
```

Save this as `slider14.html`. On the page, we have the `<audio>` tag, which has its `src` attribute set to a copyright-free audio clip hosted on Wikipedia. We also have the empty container element for our volume control.

This example uses a hosted OGG file as the source for the audio player. The OGG codec is supported in Firefox 3.5+, Chrome 3+, and Opera 11+.

In the script we first select the `<audio>` element using the standard jQuery syntax and retrieve the actual DOM element from the jQuery object, so that we can call methods from the `<audio>` API.

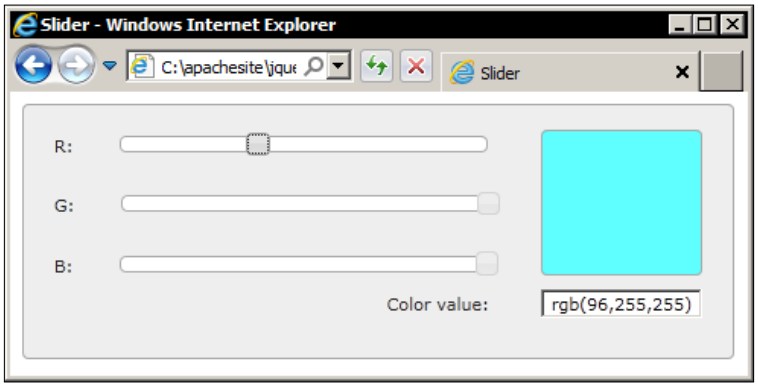
Next, we define the configuration object for our slider; we set the initial minimum and maximum values, and set the slider to `vertical`. We then add a handler for the `change` event, which is used to change the volume of the currently playing audio track, using the `volume` property and `play()` method. Whenever the value of the slider is changed, we get the new value and convert it to the required format for the `volume` option, by dividing it by `10`.

Once our variables are defined, we set the volume of the audio clip and begin playing the clip immediately with the `play()` method.

When we run this example in a supporting browser, the only thing visible on the page will be the volume slider, but we should also be able to hear the audio clip. Whenever the slider handle is moved, the volume of the clip should increase or decrease.

A color slider

A fun implementation of the slider widget, which could be very useful in certain applications, is the color slider. Let's put what we've learned about this widget into practice to produce a basic color choosing tool. The following screenshot shows the page that we'll be making:



In a new file in your text editor, create the following page:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet"
      href="css/smoothness/jquery-ui-1.8.9.custom.css">
    <link rel="stylesheet" href="css/sliderTheme3.css">
    <title>Slider</title>
  </head>
  <body>
    <div id="container" class="ui-widget ui-corner-all ui-
      widget-content ui-helper-clearfix">
      <label>R:</label>
      <div id="rSlider"></div><br>
      <label>G:</label>
      <div id="gSlider"></div><br>
      <label>B:</label>
      <div id="bSlider"></div>
      <div id="colorBox" class="ui-corner-all ui-widget-
        content">
      </div>
      <input id="output" type="text" value="rgb(255,255,255)">
      <label for="output" id="outputLabel">Color value:</label>
```

```
    </div>
    <script src="development-bundle/jquery-1.4.4.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.core.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.widget.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.mouse.js">
    </script>
    <script src="development-bundle/ui/jquery.ui.slider.js">
    </script>
    <script>
      $(function(){
        var sliderOpts = {
          min:0,
          max: 255,
          value: 255,
          slide: function() {
            var r = $("#rSlider").slider("value"),
            g = $("#gSlider").slider("value"),
            b = $("#bSlider").slider("value");
            var rgbString = ["rgb(", r, ",", g, ",", b, ")"]
             .join("");
            $("#colorBox").css({
              backgroundColor: rgbString
            });
            $("#output").val(rgbString);
          }
        };
        $("#rSlider, #gSlider, #bSlider").slider(sliderOpts);
      })(jQuery);
    </script>
  </body>
</html>
```

Save this as `slider15.html`. The page itself is simple enough. We've got some elements used primarily for displaying the different components of the color slider, as well as the individual container elements, which will be transformed into slider widgets. We use three sliders for our color chooser, one for each RGB channel.

We'll need some CSS as well to complete the overall appearance of our widget. In a new page in your text editor, create the following style sheet:

```
#container {
  width:426px; height:146px; padding:20px 20px 0;
  position:relative; font-size:11px; background:#eee;
}
#container label {
  float:left; text-align:right; margin:0 30px 26px 0;
  clear:left;
}
.ui-slider { width:240px; float:left; }
.ui-slider-handle { width:15px; height:27px; }
#colorBox {
  width:104px; height:94px; float:right; margin:-83px 0 0 0;
  background:#fff;
}
#container #outputLabel {
  float:right; margin:-14px 34px 0 0;
}
#output {
  width:100px; text-align:center; float:right; clear:both;
  margin-top:-17px;
}
```

Save this as `colorSliderTheme.css` in the `css` folder.

In our script, we give various elements such as the container and color box elements, classnames from the CSS framework, so that we can take advantage of effects like the rounded corners, and so that we can cut down on the amount of CSS we need to write ourselves.

The JavaScript is as simple as the underlying markup. We first set the configuration object. As RGB color values range from `0` to `255`, we set the `max` option to `255` and the `value` option to `255` as well, so that the widget handles start in the correct location (the color box will have a white background on page load).

The `slide` callback is where the action happens. Every time a handle is moved, we update each of the `r`, `g`, and `b` variables, by using the `value` method in `getter` mode, and then construct a new RGB string from the values of our variables. This is necessary, as we can't pass the variables directly into jQuery's `css()` method. We also update the value in the `<input>` field.

When we run the example, we should find that everything works as expected. As soon as we start moving any of the slider handles, the color box begins to change color and the `<input>` updates.

The `slide` event is a potentially intensive event that may cause issues in older browsers or on slow computers. Care should be taken when used in a production environment.

Summary

In this chapter, we looked at the slider widget and saw how quickly and easily it can be put on the page. It requires minimal underlying markup and just a single line of code to initialize.

We looked at the different options that we can set, in order to control how the slider behaves and how it is configured once it's initialized. It can be fine-tuned to suit a range of implementations.

We also saw the rich event model that can easily be hooked into, and reacted to, with up to four separate callback functions. This allows us to execute code at important times, during an interaction.

Finally, we looked at the range of methods that can be used to programmatically interact with the slider, including methods for setting the value of the handle(s), or getting and setting configuration options after initialization.

These options and methods turn the widget into a useful and highly functional interface tool that adds an excellent level of interactivity to any page.

In the next chapter, we look at the datepicker widget, which has the biggest, most feature-packed API of any widget in the library, and includes full internationalization.