PHP and script.aculo.us Web 2.0 Application Interfaces

Sridhar Rao

Chapter No. 8

"Slider for Dynamic Applications using script.aculo.us"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.8 "Slider for Dynamic Applications using script.aculo.us"

A synopsis of the book's content

Information on where to buy this book

About the Author

Sridhar Rao has been learning, working, and developing web applications from the time he was first introduced to the Web. The very idea of reaching out to the masses and bringing change in the behavior of the users through web applications excites him the most.

Most of his work has been in PHP, MySQL, and JavaScript. He has worked with some of the leading technology and service companies in his IT career.

Sridhar currently works for the world's leading database and enterprise company. He holds an engineering degree in Information Technology and is based in Bangalore, India.

A book is not the work of an individual. I would like to thank my family and friends for their encouragement and support. I would like to thank the whole team of Packt who not only helped me when things were difficult, but also believed in this project. Special mention goes to James Lumsden, Nikhil Bangera, Rajashree Hamine, Bhupali Khule, Hithesh Uchil, and Navya Diwakar for their extra efforts and patience.

PHP and script.aculo.us Web 2.0 Application Interfaces

Let me start by thanking the whole script.aculo.us community, which is pushing the limits of creativity through JavaScript.

This book is a humble attempt to help developers to quickly get on board and make their web applications AJAXified using Prototype and script.aculo.us. We have used PHP and MySQL as our server-side artillery to spread love among the PHP and MySQL developers and community as a whole for script.aculo.us.

Prototype library has been covered in depth and features have been explained in a way that would not only help a beginner but also amaze gurus. The script.aculo.us library has been fully explored with the help of snippets, codes, and examples.

Exclusive hands-on examples have been provided that will act as a reference guide whenever needed.

Towards the end of the book we go on to build three web applications from scratch.

"If Prototype is giving our web applications powerful performance, script.aclo.us is making them look functionally beautiful."

What This Book Covers

Chapter 1 Kick-starts our script.aculo.us journey. We will explore the overview of the script.aculo.us library, real-world usage, and a quick example.

In *Chapter 2* we will learn about the powerful Prototype library. We will explore various features like DOM, AJAX, event handling, and helper functions.

Chapter 3 gets us started with PHP and MySQL in building our complete Login Management System, getting AJAX into the picture, and create our own Tag Cloud.

In *Chapter 4* we will learn with the help of hands-on examples, how to add multimedia and effects to web applications using script.aculo.us.

In *Chapter 5* we will learn to make simple, clean, and beautiful user interfaces using drag and drop. Drag everything and drop something.

In *Chapter 6* we will learn how to use InPlaceEditor and InPlaceCollection for editing on the fl y.

Chapter 7 explores yet another 2.0 feature called autocompletion to create more robust and engaging applications.

In *Chapter 8* we will learn the hands-on examples with different types of sliders and how to integrate it into our web applications.

Chapter 9 is our reference guide for all the script.aculo.us features in one go.

In *Chapter 10* we will learn how to build our own tadalist application from scratch to live.

In *Chapter 11* we will build your own social bookmarking application from scratch to live.

In *Chapter 12* we will learn how to build a new design for a 2.0 shopping site from scratch to live.

Chapter 13 explains the build modules required to implement 43 things, 43 people, and 43 places from scratch to live.

8

Slider for Dynamic Applications using script.aculo.us

Handling, processing, and representing data in the 2.0 era of web applications has become so crucial that designers and programmers are working towards new ways of improving the user interface experience.

Slider is one such killer concept, using which the user can represent and handle data easily.

A slider, according to the dictionary, stands for "the one that slides". Yes, a slider in the web application context stands for holding and sliding values from a fixed given range, or even from an array of values.

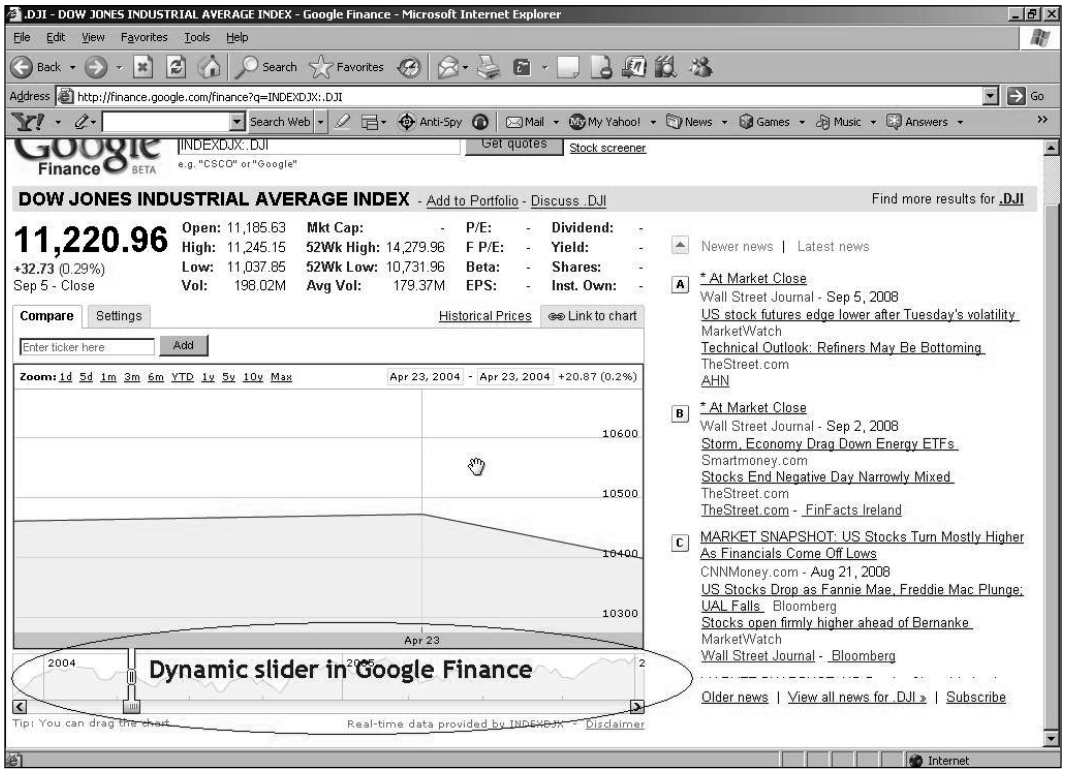
The slider is really useful and worthy in places where the user needs to slide through a lot of values and also the application needs to respond to those values and changes.

Some of the key topics we will cover in this chapter are:

- First steps with the script.aculo.us slider
- Types of the slider
- Code usage for the slider
- Tips and tricks with the slider
- Hands-on example with vertical and horizontal slider

Before we start exploring the slider, let me try to give you a complete picture of its functionality with a simple example.

Google Finance uses a horizontal slider, showing the price at a given day, month, and year. Although this particular module is built in Flash, we can build a similar module using the script.aculo.us slider too. To understand the concept and how it works, look at the following screenshot:



Now that we have a clear understanding of what the slider is and how it appears in user interface, let's get started!

First steps with slider

As just explained, a slider can handle a single value or a set of values. It's important to understand at this point of time that unlike other features of script.aculo.us, a slider is used in very niche applications for a specific functionality.

The slider is not just mere functionality, but is the behavior of the users and the application.

A typical constructor syntax definition for the slider is shown as follows:

```
new Control.Slider(handle, track [ , options ] );
```

Track mostly represents the `<div>` element. Handle represents the element inside the track and, as usual, a large number of options for us to fully customize our slider.

For now, we will focus on understanding the concepts and fundamentals of the slider. We will surely have fun playing with code in our *Code usage for the slider* section.

Parameters for the slider definition

In this section we will look at the parameters required to define the slider constructor:

- `track` in a slider represents a range
- `handle` in a slider represents the sliding along the track, that is, within a particular range and holding the current value
- `options` in a slider are provided to fully customize our slider's look and feel as well as functionality

It's time to put the theory into action. We need the appropriate markup for working with the slider. We have `<div>` for the `track` and one `<div>` for each `handle`. The resulting code should look like the snippet shown as follows:

```
<div id="track"><div id="handle1"></div></div>
```

It is possible to have multiple handles inside a single track. The following code snippet is a simple example:

```
<div id="track"><div id="handle1"></div>
<div id="handle2"></div></div>
```

Options with the slider

Like all the wonderful features of script.aculo.us, the slider too comes with a large number of options that allow us to create multiple behaviours for the slider. They are:

- `Axis`: This defines the orientation of the slider. The direction of movement could be horizontal or vertical. By default it is horizontal.
- `Increment`: This defines the relation between value and pixels.
- `Maximum`: This is the maximum value set for the slider to move to. While using a vertical slider from top-to-bottom, the bottom most value will be the maximum. And for a horizontal slider from left-to-right, the right most value will be the maximum value.

- `Minimum`: This is the minimum value set for the slider to move to. While using a vertical slider from top-to-bottom, the top most value will be the minimum. And for a horizontal slider from left-to-right, the left most value will be the minimum value approach for horizontal slider.
- `Range`: This is the fixed bandwidth allowed for the values. Define the minimum and maximum values.
- `Values`: Instead of a range, pass a set of values as an array.
- `SliderValue`: This sets the initial value of the slider. If not set, will take the extreme value of the slide as the default value.
- `Disabled`: As the name suggests, this disables the slider functionality.

Some of the functions offered by the slider are:

- `setValue`: This will set the value of the slider directly and move it to the value position.
- `setDisabled`: This defines that the slider is disabled at runtime.
- `setEnabled`: This can enable the slider at runtime.

Some of the callbacks supported by the slider are:

- `onSlide`: This is initiated on every slide movement. The called function would get the "current" slider value as parameter.
- `onChange`: Whenever the value of the slider is changed, the called function is invoked. The value can change due to the slider movement or by passing the setValue function.

Types of slider

script.aculo.us provides us the flexibility and comfort of two different orientations for the slider:

- Vertical slider
- Horizontal slider

Vertical slider

When the axis orientation of a slider is defined as vertical, the slider becomes and acts as a vertical slider.

Horizontal slider

When the axis orientation of a slider is defined as horizontal, the slider becomes and acts as a horizontal slider.

So let's get our hands dirty with code and start defining the constructors for horizontal and vertical slider with options. Trust me this will be fun.

Code usage for the slider

As a developer, I am sure you must have got a little bored reading *only* explanation. But hey hang on, we are getting into code!

Let's start with our HTML code and then the basic constructor definition of the slider.

The HTML code snippet is shown as follows:

```
<div id="track"><div id="handle1"></div>div>
```

Here, we have defined our track and handle as `<div>` elements.

The handle element should be placed inside the `track` element.

Good. So let's define the constructor for the slider here:

```
new Control.Slider('handle1', 'track1');
```

That's it! No, wait. We are missing something. Although the code is perfect, when we fire it up in the browser we can't see anything. That's because we need to style it.

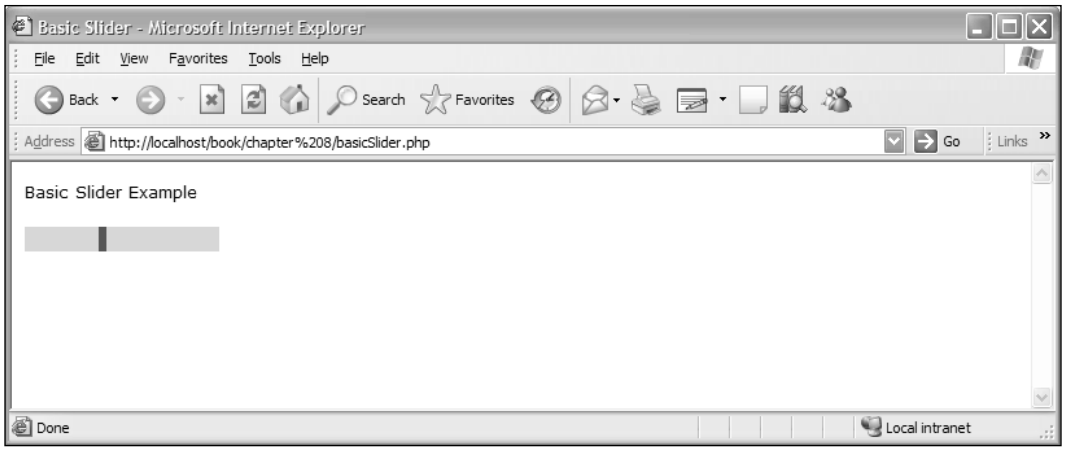
The complete code with CSS is shown as follows:

```
<script type="text/javascript">
   window.onload = function() {
       new Control.Slider('handle1', 'track1'
       );
}
</script>
<style type="text/css">
h4{ font: 13px verdana }
#track1 {
   background-color:#BCE6D6;
   height: 1em;
   width: 150px;
}
```

```
#handle1 {
   background-color:#30679B;
   height: 1em;
   width: 6px;
}
</style>

<body>
<h4>Basic Slider Example</h4>
<div id="track1">
   <div id="handle1"></div>
</div>
```

And the resulting output is shown in the following screenshot:



That's the most basic slider we created. And I am sure you are not content with that. We need to explore more.

Code usage for the vertical slider

Moving on, we will now create a vertical slider and add some options to enhance our slider feature.

Most of the code remains from the above example. We will focus on the required changes to be made in the above code.

As mentioned in the explanation above, we need to define the axis orientation as `vertical` in our options to make a slider vertical.

```
axis: 'vertical'
```

So, the new constructor looks like the snippet shown as follows:

```
window.onload = function() {
  new Control.Slider('handle1', 'track1',
    {
   axis:'vertical'
    }
   );
}
```

And yes, since we are trying to make our slider vertical we need to change the CSS properties of height. The new CSS code will look like the following snippet:

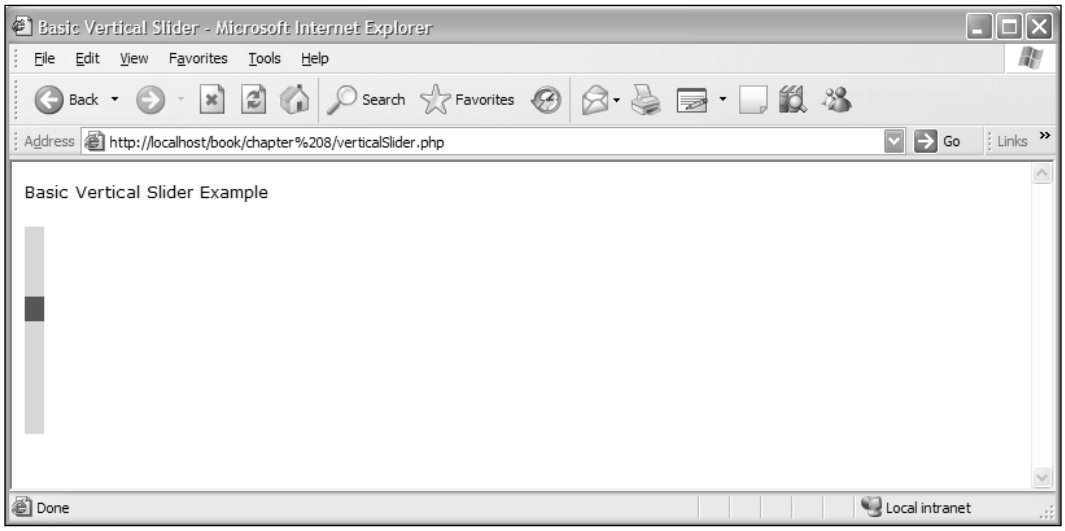
```
#track1 {
   background-color:#BCE6D6;
   height: 10em;
   width: 15px;
}
#handle1 {
   background-color:#30679B;
   height: 1em;
   width: 15px;
}
```

So, the final script for the vertical slider is shown as follows:

```
<script type="text/javascript">
   window.onload = function() {
       new Control.Slider('handle1', 'track1',
       {
   axis:'vertical'
       }
       );
}
</script>
<style type="text/css">
h4{ font: 13px verdana }
#track1 {
   background-color:#BCE6D6;
   height: 10em;
   width: 15px;
}
#handle1 {
```

```
    background-color:#30679B;
    height: 1em;
    width: 15px;
}
</style>
</head>
<body>
<h4>Basic Vertical Slider Example</h4>
<div id="track1">
    <div id="handle1"></div>
</div>
```

And, the beautiful vertical slider is here! Check out the following screenshot:



Code usage for the horizontal slider

We have seen how to create a vertical slider. We want you to have a wild guess of how to make a horizontal slider. Let me give you two hints:

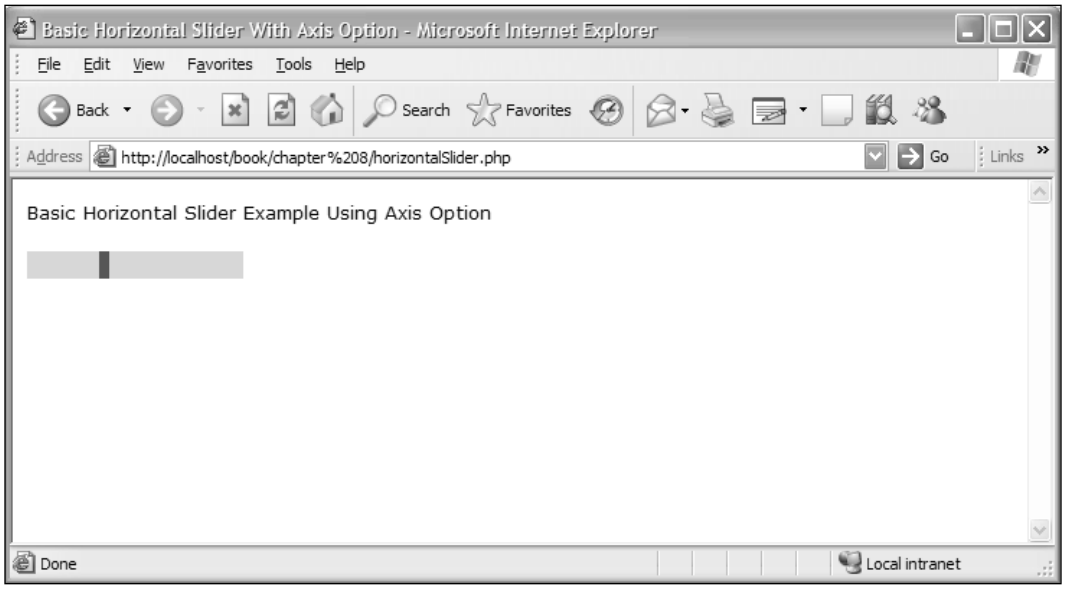
- We don't have to struggle to make a slider horizontal. It's the default axis option.
- We can make a horizontal slider by passing the "horizontal" option to axis.

Which one would you prefer?

I am not going to give you code for this one though. But yes, I will guide you for doing the same. The code will be given in the next chapter.

We have already created one horizontal slider in the *Code usage for the horizontal slider* section. That was one approach. Now try changing the axis option to `horizontal` in the above code for the vertical slider.

You may also need to change some CSS properties for height and width, and I am sure you would love doing them. It's so much fun! After you make changes to the height and width parameters of the CSS properties, the screenshot of slider should look like the following:



Code usage for sliders with options

We are now done with the most important part of the slider: the implementation of the slider in our applications.

But wait, we need the slider to suit our applications, right? So let's customize our slider with options.

We have mentioned earlier that `track` is the range of values. So let's first define the range for our slider.

```
window.onload = function() {
       new Control.Slider('handle1', 'track1',
       {
   axis:'vertical',
   range:$R(1,100)
   }
```

The range option uses the Prototypes' `objectRange` instance. Hence, we declare it using

```
$R (minimum, Maximum).
```

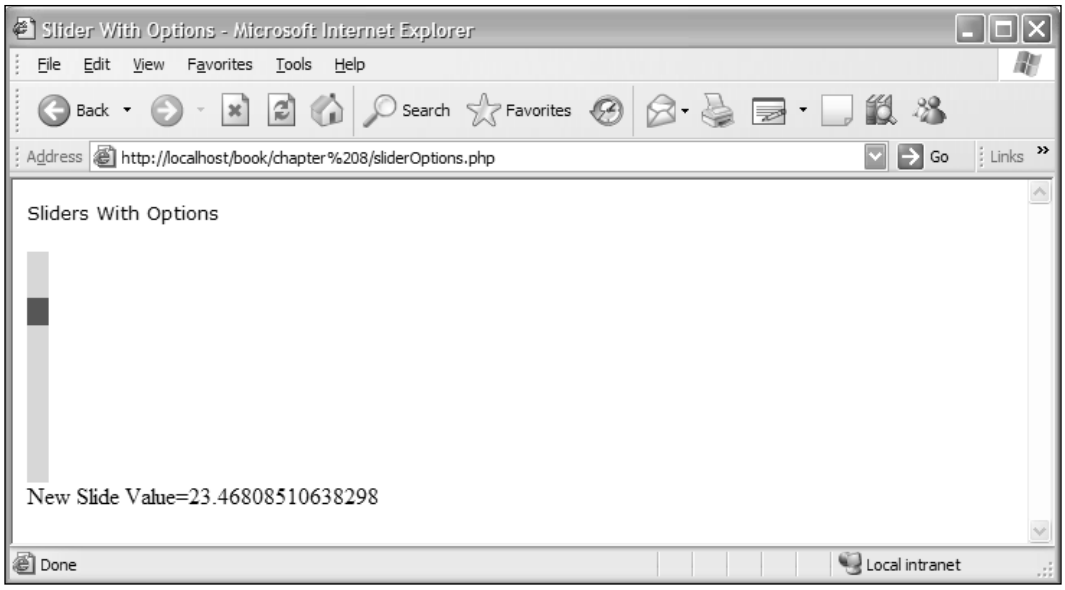
Everything looks neat until here. Let's add some more options to our constructor, `onSlide()`.

Using the `onSlide()` callback every time, we drag the slider and the callback is invoked. The default parameter passed to `onSlide()` is the current slider value.

```
window.onload = function() {
       new Control.Slider('handle1', 'track1',
       {
   axis:'vertical',
   range:$R(1,100),
   onSlide: function(v) { $('value1').innerHTML = "New Slide
                                                   Value="+v;}
   }
```

We have added a div called `value1` in our HTML code. On dragging the slider, we will update the `value1` with the current slider value.

OK, so let's see what happened to our slider uptill now. Check out the following screenshot:



Impressed? And, we are not done yet. Let's add more options to the slider now.

You may ask me, what if the slider in the application needs to be at a particular value by default? And I will say use the `sliderValue` option. Let's make our slider value `10` by default. Here is the snippet for the same:

```
window.onload = function() {
       new Control.Slider('handle1', 'track1',
       {
         axis:'vertical',
         range:$R(1,100),
         sliderValue: 10,
         onSlide: function(v) { $('value1').innerHTML =
                           "New Slide Value="+v;}
       }
}
```

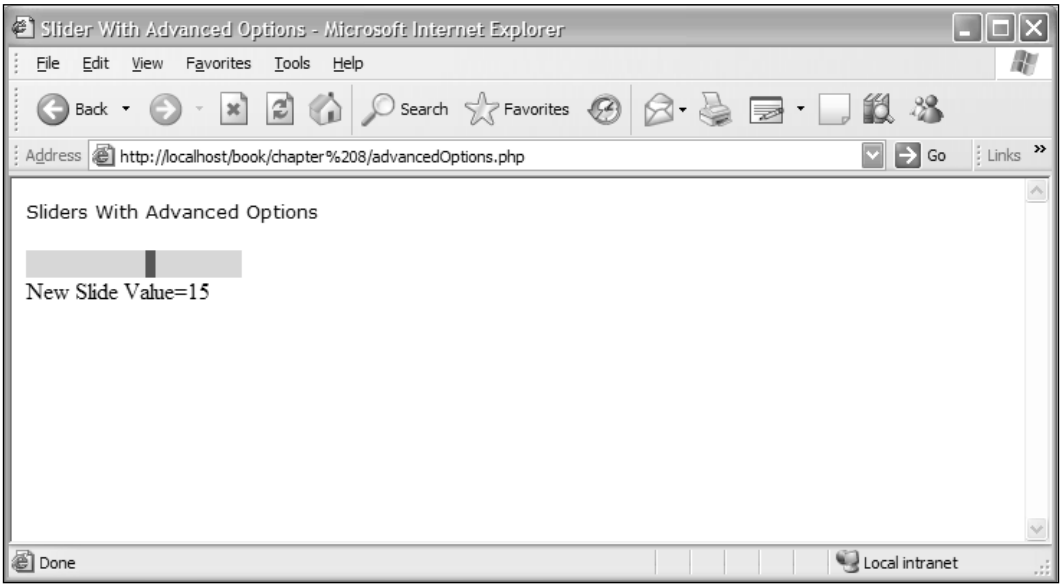
And, you should see the slider value at `10` when you run the code.

Now your dear friend will ask, what if we don't want to give the range, but we need to pass the fixed set of values? And you proudly say, use the `values` option.

Check out the usage of the `values` options in the constructor.

```
window.onload = function() {
 new Control.Slider('handle1', 'track1',
 {
    range:$R(1,25),
    values:[1, 5,10,15,20,25],
    onSlide:function(v){ $('value1').innerHTML = "New Slide
                                                  Value="+v;}
  }
 );
}
```

We have added a set of values in the array form and passed it to our constructor. Let's see what it looks like.



Tips and tricks with the slider

After covering all the aspects of the slider feature, here is a list of simple tips and tricks which we can make use of in our applications with ease.

Reading the current value of the slider

script.aculo.us "genie" provides us with two callbacks for the slider to read the current value of the slider. They are:

- `onSlide`
- `onChange`

Both these callbacks are used as a part of `options` in the slider.

`onSlide` contains the current sliding value while the drag is on. The callback syntax is shown as follows:

```
onSlide: function(value) {
// do something with the value while sliding. Write or Edit the
//value of current slider value while sliding
}
```

`onChange` callback will contain the value of the slider while the "sliding" or the drag event ends. After the drag is completed and if the value of the slider has changed then the `onChange` function will be called. For example, if the slider's current value is set to `10` and after sliding we change it to `15`, then the `onChange` callback will be fired. The callback syntax is shown as follows:

```
onChange: function(value){
   // do anything with the "changed" and current value
}
```

Multiple handles in the slider

Now, a thought comes to our mind at this point: Is it possible for us to have two handles in one track?

And, the mighty script.aculo.us library says yes!

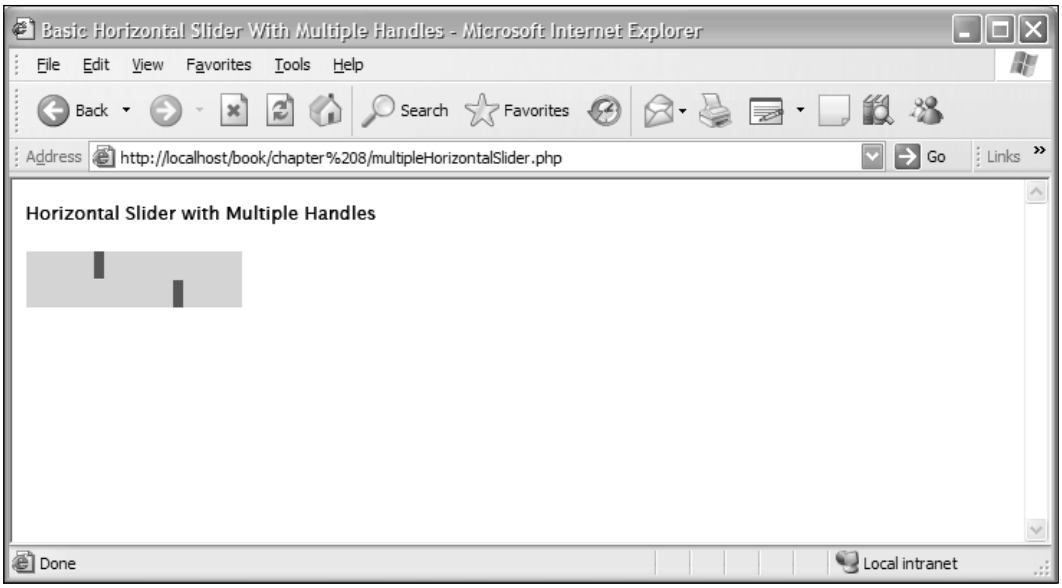
Check out the following code snippet and screenshot for a quick glance of having two handles in one track:

```
HTML code
<div id="track1">
   <div id="handle1"></div>
   <div id="handle2"></div>
</div>
```

JavaScript code for the same:

```
window.onload = function() {
       new Control.Slider(['handle1','handle2'] , 'track1'
       );
}
```

Now, check out the resulting screenshot having two handles and one track:



The same can also be applied for the vertical slider too.

Disabling the slider

We can disable our slider element using the option: `disabled`. We need to pass `true` to set the element in the disabled state. By default the value is set to `false`.

Our constructor definition would look like the code snippet shown as follows:

```
window.onload = function() {
 new Control.Slider('handle1', 'track1',
 {
    range:$R(1,25),
    values:[1, 5,10,15,20,25],
     disabled:true,
```

```
        onSlide:function(v){ $('value1').innerHTML = "New Slide
                                                      Value="+v;}
    }
   );
  }
```

The `disabled` option will initially make the element's state disabled, and we can change this state using `setDisabled`.

Enabling the slider

As we can disable our slider element using the `disabled` option, we can also enable the element using the same option by passing the value as `false`.

Our constructor definition would look like the code snippet shown as follows:

```
window.onload = function() {
 new Control.Slider('handle1', 'track1',
 {
    range:$R(1,25),
    values:[1, 5,10,15,20,25],
     disabled:false,
     onSlide:function(v){ $('value1').innerHTML = "New Slide
                                                   Value="+v;}
  }
 );
}
```

By default the value of the `disabled` option is `false`. The elements are enabled, and we can change the state using `setEnabled`.

Hands-on example: Using vertical and horizontal slider

Now that we have worked with vertical and horizontal slider, wouldn't it be a great idea to see both types of the slider on the same page? Yes indeed.

Let's get started.

At a very basic level, we can change the `Axis` option of slider and we can get either the horizontal or vertical slider.

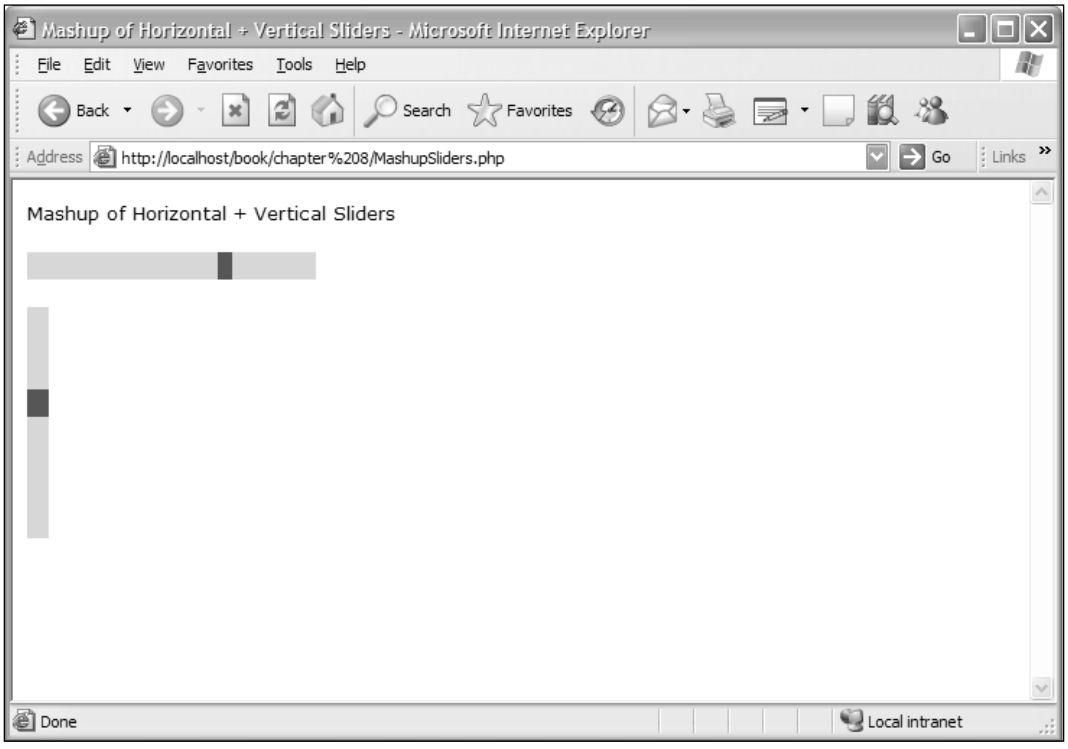
So now we will have two slider types on one page, and the only difference is in the axis orientation.

We need to create two tracks and the respective handles for the slider `<div>`s. The HTML part of the code is given as follows:

```
<h4>Mashup of Horizontal + Vertical Sliders</h4>
<div id="track1" class="track">
   <div id="handle1" class="handle"></div>
</div>
<div id="track2">
   <div id="handle2"></div>
</div>
```

This code is pretty simple. We have created a `<div>` as `track1` and its respective inner `<div>` to hold the value as `handle1`. Similarly, we have created one more slider `<div>` as `track2` and its handle as `handle2`.

After a bit of trendy dressing up of, and applying make-up to, the CSS, the basic slider looks like the following screenshot:



The CSS code is given here:

```
h4{ font: 13px verdana }
#track1 {
   background-color:#BCE6D6;
   height: 1em;
   width: 200px;
}
#handle1 {
   background-color:#30679B;
   height: 0.5em;
   width: 10px;
}
#track2 {
    background-color:#BCE6D6;
    height: 10em;
    width: 15px;
 }
 #handle2 {
    background-color:#30679B;
    height: 1em;
    width: 15px;
}
#sliding {
font: 13px verdana;
}
#changed {
font: 13px verdana;
}
```

OK, so now we have our slider in our page. Wait, we are missing something. Are you wondering where is the scripting and functionality?

Before that, let's add two divs, which will help us view the current values using the `onChange` and `onSlide` callbacks.

```
<p id="sliding"></p>
<p id="changed"></p>
```

Now let's first add script.aculo.us power to `track1`, our first slider.

```
new Control.Slider('handle1', 'track1',
        {
             range: $R(1,50),
             values: [1,5,15,25,35,45,50],
             sliderValue: 1,
             onChange: function(value){
```

```
            $('changed').innerHTML = 'Changed Value : '+value;
        },
        onSlide: function(value) {
            $('sliding').innerHTML = 'Sliding Value: '+value;
        }
    } );
```

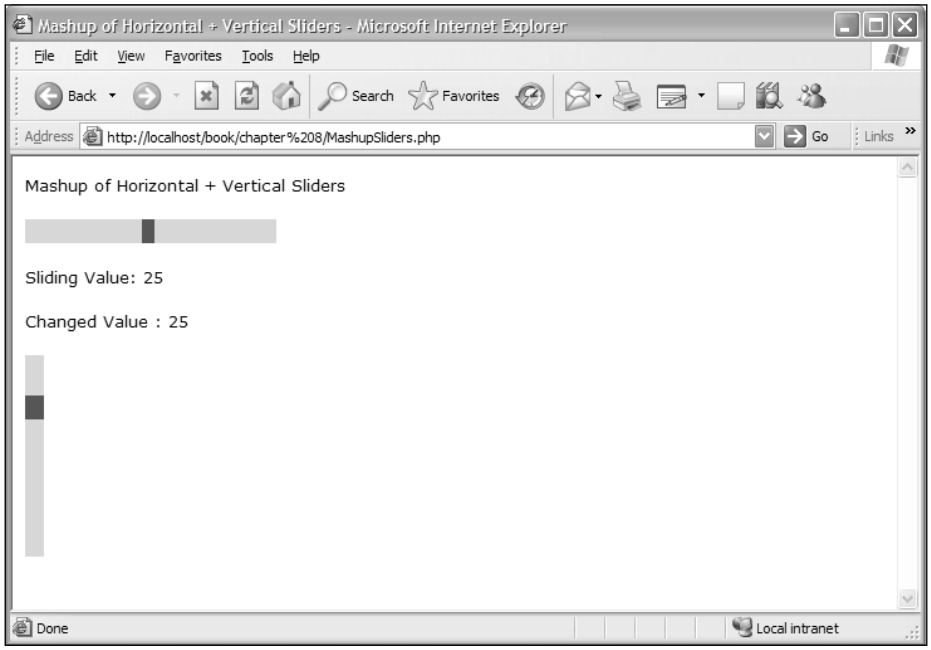
Let's take a closer look at the above code and see what is happening.

We have defined `range`, `values`, and `sliderValue` options for the slider. We have also added two callbacks `onChange` and `onSlide`. As I mentioned earlier, these callbacks get the *current* value of the slider as a parameter.

Hence, we are reading values from both the callbacks and updating the divs' *sliding* and *changed* when the event occurs.

Also, since we did not exclusively mention the axis definition, the default is `horizontal`.

So, the application now looks like the following screenshot:



Remember, the values will *only* be updated if we move the horizontal slider. And, nothing happens if we slide through the vertical slider. We have not yet defined the functionality for the vertical slider. Let's do it now.

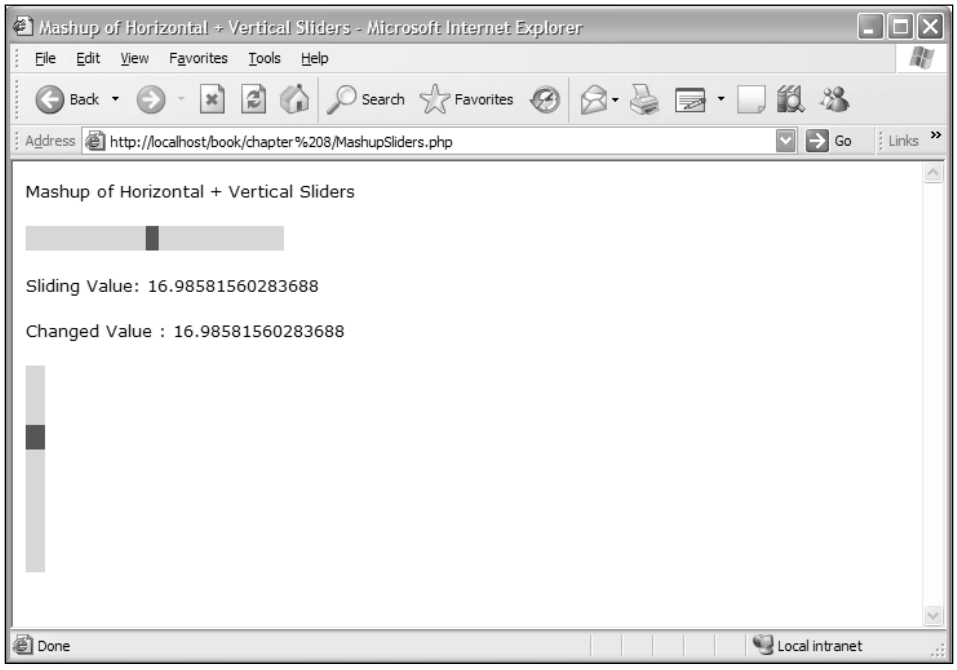
The code for the vertical slider will also remain mostly similar with only difference of `axis` orientation.

```
new Control.Slider('handle2', 'track2',
        {
             range: $R(1,50),
             axis:'vertical',
             sliderValue: 1,
             onChange: function(value){
                   $('changed').innerHTML =
                     'Changed Value : '+value;
             },
             onSlide: function(value) {
                   $('sliding').innerHTML =
                     'Sliding Value: '+value;
            }
         } );
```

You can notice the fact that the callback definition remains the same for the vertical slider as well.

So now when we move the vertical slider, the value gets updated in the *sliding* and *changed* `<div>`s. They get updated with the current value.

The complete module with both horizontal and vertical slider is shown as follows:



Summary

In this chapter we have learned and explored the following topics:

- Introduction to the slider using script.aculo.us
- Explanation of the slider
- Different types of the slider
- Options provided with the slider
- Code usage for the slider and options
- Tips and tricks with slider
- Hands-on example for the horizontal and vertical slider

So far we have learned all the features of script.aculo.us in detail. We have also worked on some hands-on examples to make us more comfortable using the features of script.aculo.us in our own applications.

But we think it would be nice to have a cheat sheet of all the features in one page. This would act as a reference for us at any point in time. We will cover all this and more in the next chapter!