At the Forge

Scriptaculous

Reuven M. Lerner

Abstract

Scriptaculous is a spectaculous set of libraries for Ajax.

Ajax is the hot new Web development paradigm that uses JavaScript to send and then handle asynchronous HTTP requests. The past few months, this column has looked into different ways to spawn and handle Ajax calls. The most complicated way was to do it ourselves, creating an XMLHttpRequest object, and then using it to send a request to the user's browser as well as to specify which JavaScript function will handle the response. Last month, we showed that we can simplify our lives greatly by using Prototype, a JavaScript library that includes many of the shortcuts and utility functions that are of use to JavaScript programmers.

The good news is that Prototype does indeed make JavaScript programming easier and more straightforward. But, one of the things people most want to do with JavaScript is create more flexible GUIs. This is especially true now that Web applications are becoming more desktop-like; users expect to have the same sense of feedback and control that they have with their nonbrowser applications.

Just as we saw with Ajax, there are ways to create and re-use these behaviors on your own. But why would you do that, when there are libraries that handle such tasks for you? Several of these are Scriptaculous, MoochiKit and Dojo. (Dojo is actually a complete top-to-bottom JavaScript library; I expect to look at it in a future installment of this column.) This month, we look at Scriptaculous, an open-source library written by Thomas Fuchs. Scriptaculous makes it easy to spruce up our HTML files without having to delve into the low-level JavaScript.

Basics of Scriptaculous

Installing Scriptaculous couldn't be easier. Download the latest version of Scriptaculous (http://script.aculo.us), and install the six included JavaScript files (in the src directory) somewhere in your Web server's document root.

Actually, installing Scriptaculous could be even easier than this. If you use a recent version of Ruby on Rails, Scriptaculous and Prototype are already installed. See the Rails documentation for a description of how to use these libraries directly, as well as from Ruby code.

Note that Scriptaculous 1.6.5, which I use in this article, requires Prototype 1.5 or above. Although Prototype 1.5 likely will be released by the time this column sees print, I currently am relying on Prototype 1.5 RC1. Thus, there might be some differences between the functionality I describe here and the final distribution.
In order to use Scriptaculous, you need to include two script tags in your HTML page to load Prototype and then Scriptaculous:

```html
<script src="/javascripts/prototype.js" type="text/javascript"></script>
<script src="/javascripts/scriptaculous.js" type="text/javascript"></script>
```

**Simple Effects**

So, what might you want to do with Scriptaculous? One of its most common uses is in the creation of visual effects. Each effect is defined as a method within the Effect object. You can create an effect by saying:

```js
new Effect.EffectName('id')
```

where EffectName is the name of the effect that you want, and id is the ID of the HTML element on which the effect will take place. For example, if we have the following headline:

```html
<h2 id="headline">The headline</h2>
```

we can make it fade by invoking:

```js
new Effect.Fade('headline');
```

Of course, it makes sense for such things to happen only when particular events occur. Listing 1 contains a simple HTML file, effects.html, with two buttons labeled appear headline and fade headline. Clicking on the first button invokes Effect.Appear on the node with an ID of headline. Note that we don't pass the node itself to Effect.Fade, but rather the ID. Effect.Fade uses Prototype's $() function to retrieve the node with that ID.

**Listing 1. effects.html**

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Special effects</title>
<script src="/javascripts/prototype.js" type="text/javascript"></script>
<script src="/javascripts/scriptaculous.js" type="text/javascript"></script>
</head>
<body>
<h2 id="headline">Welcome</h2>
<p>Welcome to the page of effects!</p>
<p><input type="button" value="Fade headline" onclick="new Effect.Fade('headline')" /></p>
<p><input type="button" value="Appear headline" onclick="new Effect.Appear('headline')" /></p>
</body>
</html>
```
To make the headline fade, we set the following event handler:

```javascript
onclick="new Effect.Fade('headline')"
```

Each of the effects has a number of settings, each of which is given a default value by Scriptaculous. To override one or more of these defaults, pass one or more of them in the invocation:

```javascript
onclick="new Effect.Fade('headline', {delay: 2, duration: 10})"
```

In some cases, such as the appear/fade duo in Listing 1, it seems a bit silly to have two buttons. After all, what happens if I click on fade twice? It would be more reasonable to have a single button that turns the headline on when it's off and vice versa. Scriptaculous supports this with the toggle effect. For example, we can remove one button and give the second one an event handler that looks like this:

```javascript
onclick="new Effect.toggle('headline', 'appear')"
```

Now, clicking on that button toggles the visibility of the headline, using appear and fade to turn the headline on and off. We can do the same thing with the blind (BlindUp and BlindDown) and slide (SlideUp and SlideDown) effects as well. We also can combine a toggle with the parameters shown earlier:

```javascript
onclick="new Effect.toggle('headline', 'appear',
{delay: 2, duration: 10})"
```

Only a few effects come in pairs. Several are useful only for removing text, as in the following:

```javascript
new Effect.Fold("headline")
```

Others are made to get the user's attention. For example, we can highlight our headline by doing this:

```javascript
new Effect.Highlight("headline")
```

## Autocomplete

Effects are certainly an important and impressive part of Scriptaculous; the library comes with many effects, and there are numerous ways to combine and invoke them. However, Scriptaculous offers much more than merely a bunch of effects. It also includes some user-level functionality JavaScript programmers might want.

One such example, popularized by Google Suggest (and its cousin in Firefox 2.0) is an autocompleting text field. Such a text field lets you enter whatever contents you want, but if what you have entered so far matches a known text string, you are offered the chance to complete it. This is similar in many ways to the combo box widget that was long popular on Windows systems, but which was unavailable to Web applications. (Some Scriptaculous users have created derivatives of the Autocompleter class that is more similar to a combo box.)

Scriptaculous comes with two different types of autocompleting text fields, differing only in how the completion list is filled. In the first case, known as Autocompleter.Local, the list of matches is set in JavaScript. The related text field, Ajax.Autocompleter, uses Ajax to retrieve a list of matches from a remote HTTP server. The two are similar enough in spirit that I demonstrate only Autocompleter.Local here for the sake of simplicity.
In order to use Autocompleter.Local, we first create a text field, just as we would do in an HTML form:

```html
<input type="text" id="distro" name="distro_text_field"
    autocomplete="off" />
```

Notice that I have included the setting `autocomplete="off"` in the above field. The autocomplete attribute, which is used only by Microsoft's Internet Explorer, is set here to off to ensure that IE users aren't unpleasantly surprised by dueling completion systems.

Next, we create a `<div>` section with an ID attribute. In addition, we set the style attribute to keep the div hidden until we modify its styling:

```html
<div id="distro_complete" style="display:none"></div>
```

Finally, we add some JavaScript that creates a new Autocompleter.Local object:

```javascript
<script type="text/javascript">
    new Autocompleter.Local('distro', 'distro_complete', 
        ['Red Hat', 'Fedora Core', 'Debian', 'Gentoo', 'Knoppix', 'Ubuntu', 'Kubuntu'], { });
</script>
```

The constructor for Autocompleter.Local takes four arguments: the ID of the text field, the ID of the div into which we'll insert completions, an array containing the completion strings and a set of options (currently empty). If you try to put this code in the `<head>` of your document, it will fail with odd errors, because the text field and div must exist before the code is executed.

By including the above in an HTML page (as in Listing 2), you set the stage for autocompletion to work. Whenever the user loads the page and types a letter into the text field, Scriptaculous waits for 0.4 seconds of inactivity. If the user isn't typing, and if the text field already contains one or more characters, Autocompleter.Local tries to find a match from the current list. If it finds one, it fills in the rest of the text field.

**Listing 2. complete.html**

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Special effects</title>
        <script src="/javascripts/prototype.js"
type="text/javascript"></script>
        <script src="/javascripts/scriptaculous.js"
type="text/javascript"></script>
    </head>

    <body>
        <h2>Enter your favorite distribution</h2>
        <p><input type="text" id="distro" name="distro_text_field"
            autocomplete="off" /></p>
        <div class="auto_complete" id="distro_complete"
            style="display:none"></div>
        <script type="text/javascript">
            new Autocompleter.Local('distro', 'distro_complete', 
                ['Red Hat', 'Fedora Core', 'Debian', 'Gentoo', 'Knoppix', 'Ubuntu', 'Kubuntu'], { });
        </script>
```
If it finds more than one (as would happen to a user typing K in our example, which matches both Kubuntu and Knoppix), the system displays a menu of options, from which the user may choose by typing or clicking.

## Conclusion

Prototype is a library aimed at the JavaScript that endeavors to solve many programmers' needs. However, Prototype's functionality extends only so far as the programmer; it doesn't offer any direct GUI improvements. Given that JavaScript is largely used to handle the GUI in a Web application, it shouldn't come as a surprise that there are several libraries built on top of Prototype to handle such tasks. Scriptaculous appears to be one of the best known of these, and in my experience, there's good reason for that.

There are several functions of Scriptaculous that we didn't get to explore this month, including drag and drop and JavaScript unit testing. This last item is probably of note even for JavaScript programmers who have no intention of creating GUI effects.

**Resources for this article:** [http://www.linuxjournal.com/article/9505](http://www.linuxjournal.com/article/9505).