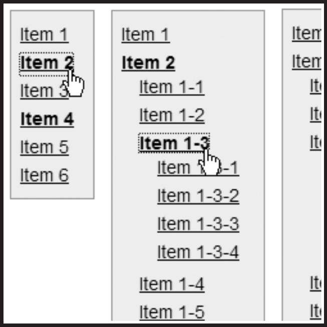
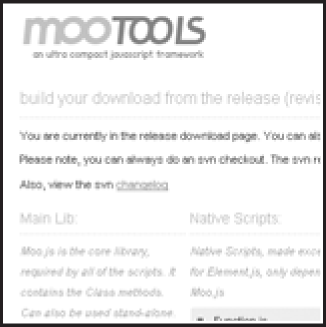
9 ADDING SPECIAL EFFECTS




mooTOOLS
an ultra compact javascript framework
build your download from the release (revis
Main Lib:
Native Scripts:

You are currently in the release download page. You can al

Please note, you can always do an svn checkout. The svn r

Also, view the svn changelog

Moo.js is the core library, required by all of the scripts. It contains the Class methods. Can also be used stand-alone.

Native Scripts, made exc for Element.js, only depen Moo.js

By Chris Heilmann

In this chapter we'll talk about using JavaScript libraries to add some extra zing to your web site. To make this as easy as possible we'll use JavaScript libraries. We've already touched on the subject of JavaScript libraries in Chapter 4, but now we'll go into more examples on how to use them. Specifically we'll pick two common JavaScript tasks and use one library at a time to fulfill them.

This chapter contains code examples. Do ***not*** *type these in yourself, as there is no need for that. You'll find these examples in the code download zip file for this chapter at* `http://www.friendsofed.com`*; simply unpack it to your local server in a folder called "libraries" or something similar, and try the examples out by double-clicking the HTML files.*

Originally we meant to add readymade scripts to put into a WordPress installation with these examples. However, seeing that some libraries are still in the process of changing in a very short period of time, it would not be helpful to give a plug-and-play solution. Instead, we want to encourage you to take a look at the library pages themselves and browse the scripts generated there. Before we go on to these examples, let's remind ourselves about the why of JavaScript and the why of libraries.

The why of JavaScript

First things first: you don't need JavaScript to have a good web site. Even more to the point: a good web site should not require JavaScript to work and make sense, but JavaScript should make some things easier for you when and if it is available.

JavaScript was invented as users demanded web sites to be more responsive. Long ago, when reading web sites meant spending a lot of money every minute and loading a page took several minutes, it was simply a necessity.

There is nothing more frustrating even today than having to load a form 10 times because you forgot to enter something or you entered something in the wrong format. Now imagine a scenario where you have to wait about 2 minutes for each reload and see your bill increasing with each of them—you have a recipe for people to bang their keyboard in frustration (no wonder the office guy doing exactly that became one of the first "funny web videos"—see `http://youtube.com/watch?v=5oTZXUI-L8s`).

Browser vendors realized that they needed to offer their impatient users something better to stay in the game and turn the information superhighway into a medium worth exploring for the mass market and not mere science fiction.

They first started with Java and applets—a technology that meant you had to load a large file first, wait for it to initialize and run, and then you'd have a richer user interface. The richer interface also meant that you didn't have to reload the page but data could be retrieved in the background.

Java didn't make it, though—the applets were large to load and very hungry when it came to system resources. First you had to wait a lot and then your computer got slow. Not really a good experience, which is why clever minds shut the door, put their thinking hats on, and came up with a concept that would revolutionize the Web: a language that was embedded in a web document or in its own file that could be loaded and executed by the browser.

Java had the problem that you needed to install the Java engine to make things run, because Java needs to be converted to code that can run on your computer. That got suspicious minds wondering why they'd need to install an outside program to improve their web experience (Flash used to suffer from the same problem). Furthermore, applets included a lot of code since they contained the user interface.

The new language wouldn't need that: the interface was already there, namely the HTML document with its links, forms, and buttons. All the language needed was the interpreter to translate JavaScript and allow it to work its magic with these interface elements, and browser vendors simply shipped that one as part of the browser code itself.

JavaScript was born—a lightweight language that allowed developers to write programs that could be downloaded in seconds, executed on your computer, and even cached (kept in a temporary file on your hard drive). You had to load the programs only once and could run them over and over again (unless you cleared your cache in between or turned caching off).

And this is what JavaScript is meant to be: a lightweight helper technology that turns web sites into more responsive interfaces. You can use it for good, but a lot of times it is used for evil—more on that later.

What JavaScript can do for you

JavaScript allows for changes to the HTML document after it has been downloaded and initially displayed by the browser. This means you can help your visitors by avoiding unnecessary page reloads and making the interface appear less cluttered. JavaScript can

- Validate user input on forms and give immediate feedback if there is an error.
- Flag changes that happen asynchronously in the background by giving messages without requiring a page reload. (Normal page requests are *synchronous*—you click something and the page reloads. *Asynchronously* means that you load data in the background and display the results when the data is loaded without leaving the main document.)
- Create modular windows and page elements that the visitor can open and close or expand and collapse.
- Allow for keyboard access to functionality, such as enabling a user to move around a menu using the cursor keys.
- Fix inconsistencies in CSS rendering.
- React to changes to the document such as the visitor clicking elements, scrolling, focusing on a text field, or dragging an item around.

You can embed JavaScript anywhere in the document enclosed by SCRIPT tags. If you add the JavaScript after the main document you can already access its elements (as they've already been sent to the browser and therefore exist) and change them. For example, the following script embedded in the HTML document hides all nested lists:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" ➥
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>JavaScript Test - Hiding Lists</title>
</head>
<body>
<ul>
  <li>Item 1</li>
  <li>Item 2
    <ul>
      <li>Nested Item 1</li>
    </ul>
  </li>
  <li>Item 3</li>
  <li>Item 4
    <ul>
      <li>Nested Item 2</li>
    </ul>
  </li>
</ul>
<script type="text/javascript">
  function hideStuff(){
    var lists = document.getElementsByTagName( 'ul' );
    for(var i = 0, j = lists.length; i < j; i++){
    if(lists[i].parentNode.nodeName.toLowerCase() === 'li'){
        lists[i].style.display = 'none';
      }
    }
  }
  hideStuff();
</script>
</body>
</html>
```

If you wanted to use this functionality on every page in the site, you'd need to copy and paste the code into each page, which mean the documents would become unnecessarily large; in addition, the JavaScript would not be cached. And it would mean that any future change to the script would have to be replicated in every document. This is why you can put the code in its own file instead and give it a file extension of .js. For example, you could call it hideNestedLinks.js and give it the following content:

```
    function hideStuff(){
      var lists = document.getElementsByTagName( 'ul' );
      for(var i = 0, j = lists.length; i < j; i++){
      if(lists[i].parentNode.nodeName.toLowerCase() === 'li'){
          lists[i].style.display = 'none';
        }
      }
    }
    hideStuff();
```

You add this script to the document by setting the src attribute of the SCRIPT element:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" ➥
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>JavaScript Test - Hiding Lists</title>
</head>
<body>
<ul>
  <li>Item 1</li>
  <li>Item 2
    <ul>
      <li>Nested Item 1</li>
    </ul>
  </li>
  <li>Item 3</li>
  <li>Item 4
    <ul>
      <li>Nested Item 2</li>
    </ul>
  </li>
</ul>
<script type="text/javascript" src=”hideNestedLinks.js”></script>
</body>
</html>
```

The most appropriate section of the document to embed scripts into is the head, though, because doing so makes it easier for maintainers to know where your JavaScript is embedded and where to change it without having to scan the whole document.

The impractical upshot of this is that browsers download scripts linked in the head before they start showing the document, which could result in a slight delay the first time the scripts are loaded (afterwards they will be cached and loaded from your hard drive, effectively eliminating the delay). Embedding the script in the head also means that it gets executed when the document—and the items you want to affect, in this case the lists—is not yet available for alteration, which will cause this example script to fail. You can avoid this problem by executing the script when the window has finished loading:

```
function hideStuff(){
    var lists = document.getElementsByTagName( 'ul' );
    for(var i = 0, j = lists.length; i < j; i++){
    if(lists[i].parentNode.nodeName.toLowerCase() === 'li'){
        lists[i].style.display = 'none';
      }
    }
  }
  window.onload = hideStuff;
```

The problem with this basic solution is that you will have only this script executing when the window has loaded. In reality you might have several scripts, all of them having to execute when the window has loaded. This is why there are several helper scripts that allow you to add your script to a queue to be executed when the window has loaded. Probably the first script to do that was Simon Willison's `addLoadEvent()` (`http://simonwillison.net/2004/May/26/addLoadEvent/`):

```
function addLoadEvent(func) {
  var oldonload = window.onload;
  if (typeof window.onload != 'function') {
    window.onload = func;
  } else {
    window.onload = function() {
      oldonload();
      func();
    }
  }
}
```

Using this function you can add the `hideStuff()` function to the window load event queue with `addLoadEvent(hideStuff)`. You can add as many functions as you want to this queue and they won't overwrite the other call. You'll find an example of that in the code archive. Probably every library out there comes with a similar function to connect functions with events.

Notice that the simple task to hide nested list elements appears quite complex in JavaScript:

1. You retrieve all the lists in the document with `getElementsByTagName()`.
2. You loop through these lists one by one with a `for` loop.
3. You test if the parent element of the current list is a `li` (and you need to use `toLowerCase()` as browsers inconsistently report the name of this parent element either as `li` or `LI`).
4. If that is the case, you set the list's `display` style property to `none`.

In CSS, all you'd have to do is to use `ul ul {display:none;}`, which is why clever JavaScript developers don't bother using a loop for this task but instead apply a CSS class name to the body of the document.

```
window.onload = function(){
  document.body.className = 'dynamic';
}
```

That way, you can define any visual changes to the document in CSS with a `body.dynamic` selector before your others—for example, `body.dynamic ul ul {display:none;}`. These changes will only be applied when JavaScript is available. There will be a slight delay before the styles get applied, though, as the `onload` event only fires when the whole document, images, and scripts have finished loading.

The why of JavaScript libraries

In the end, all browser vendors agreed that JavaScript is a great thing to have and that people should use it. What was missing for browser vendors was a selling point—a unique identifier why one browser would be better than the other one, and this is when the whole JavaScript thing went awry. Instead of following a common standard, every browser vendor came up with its own implementations and interfaces to the language to gain an advantage over the competitor.

The big competition was between Microsoft and Netscape, with Microsoft offering a lot of options that were Windows-specific (as IE is part of the Windows operating system and not a standalone application) and Netscape offering the `layer` element, which allowed for embedding other documents into the current one and positioning or animating the content. This part of web development history—known as the browser wars—still affects us now, partly because you will find a lot of scripts on the Web that were developed during that time that took only these two browsers into consideration and break in more modern browsers. If you find a `document.all` or `document.layers` in a script, you should be very suspicious.

These days most modern browsers support the W3C standards, at least to a large degree, which means you don't need to know that many browser-specific extras any longer. However, developing JavaScript is still a tricky subject. Browser bugs and JavaScript architecture limitations mean you need to know a lot of different environments your scripts will be executed in and all their faults and problems. This is where JavaScript libraries come in.

Libraries are collections of functions that take away the random element and provide you with shorter and more precise methods that do all the "browser normalization" for you. In other words, you don't need to know the problems of differing support between browsers and JavaScript itself as the developers of the libraries took on this job for you.

These are the pragmatic libraries that allow you to create web applications and dynamically enhanced web sites that work in different browsers on different operating systems without you having to know the twists and turns a developer needs to take to make these do their bidding.

There are several issues with JavaScript itself, especially with the DOM interface. For example, there is an `insertBefore()` *method to add something before something else, but there is no* `insertAfter()`*. You also cannot change the text in any given element using a simple construct such as* `element.text`*; instead you need to modify* `element.firstChild.nodeValue` *to change the text content of the element when it already has some content or* `element.appendChild(document.createTextNode('text'))` *when it doesn't. Truly not one of the most convenient interfaces.*

A different kind of library goes beyond that and does more than fix browser inconsistencies and provide shorter methods to achieve a certain goal. These libraries try to enhance or replace the language itself and use a different syntax. For example, they could be mimicking the behavior and style of higher programming languages like Java, C#, or Perl. Or perhaps the goal is to make it easier for nontechnical web designers to use JavaScript by, for example, providing methods that allow you to retrieve page elements by CSS selector or other means of retrieving nodes in a document such as XPath (`http://www.w3.org/TR/xpath`).

In essence, using JavaScript libraries is a good idea if you don't want to learn about browser bugs and differences, or if you want to be able to do something quickly without having to know a lot of JavaScript yourself.

The dangers of JavaScript libraries

There is a flipside to the benefits JavaScript libraries bring with them. Probably the biggest problem is that you rely on a third party and their skills to achieve a goal. If a certain part of a library causes issues in a new browser coming out, you won't know how to fix the problem. You'll have to make sure that you keep up-to-date with the library itself and upload any patches or upgrades as they get released. It also means that the library developers must offer upgrades and patches in the future.

Depending on the quality of the library and the dedication of its developers, the library may not be applicable to your site any longer if it becomes bigger or gets a wider audience of users with different browsers and needs. It is good to know which browsers the library code supports, and to what degree, in case people contact you about bugs on your site. Make sure that the library has a dedicated developer team, good documentation, and a community to ask for advice. A lot of one-man-show libraries that look great at first can become a nuisance when the library developer becomes too busy delivering more lucrative jobs than releasing a free JavaScript library.

The other big issue with libraries is that they cloak the code. Any library function will result in native JavaScript code under the hood as this is the only way it can be executed. This is especially the case when it comes to libraries that reinvent JavaScript syntax or offer "get elements by XPath or CSS selectors" shortcut methods. These methods in themselves are not a real problem, but when you start using them inside a loop it is quite interesting to see how much work the JavaScript parser really has to do. Take a pseudo-command like this, for example:

```
var active = getElementsByCssSelector('div ul li a.active');
```

Translated to standard JavaScript, this will result in the following code:

```
var active = [];
var divs = document.getElementsByTagName('div');
for(var i = 0; i<divs.length; i++){
  var uls = divs[i].getElementsByTagName('ul');
  for(var j = 0; j<uls.length; j++){
    var lis = uls[j].getElementsByTagName('li');
    for(var k = 0; k<lis.length; k++){
      var as  =  lis[k].getElementsByTagName('a');
      for(var l = 0; l<as.length; l++){
        if(as[l].className === 'active'){
          active.push(as[l]);
        }
      }
    }
  }
}
```

Every `for` loop in JavaScript takes a lot of time and resources (as in memory). When you nest loops you multiply this issue, and depending on how large the document and how complex the CSS selector is, you could end up with a really slow web site—although it has already loaded.

This is a very basic way of achieving this functionality, and there are cleverer ways using recursive functions. However, the underlying problem stays the same—you simulate in JavaScript what the CSS parser of the browser was built for.

Having easier access to the document and easier ways to manipulate it is a great thing. However, what you need to be aware of is that under the hood you expect a lot of the browser and the JavaScript parser. In the end, successful web sites are fast web sites and what helps your visitors is much more important than what is handy or easy for you as the developer.

There is another aspect of JavaScript that you should be aware of: avoiding the trap of creating effects for the sake of having effects.

Fighting the temptation

Dynamic effects on web sites are great. A panel that slides in and out smoothly, text that fades in, and a photo frame that grows smoothly before showing the photo are fun things to see and give the impression of a much more sophisticated user interface than a web page that simply shows and hides parts of it.

There is such a thing as overkill, though. There is also a thing called "not wanting to wait." If you ever sat through a presentation of a bad public speaker who tried to make up for this by applying a lot of transition and animation effects to their PowerPoint (or Keynote) slides, you know what we mean.

If you want to use animation and dynamic JavaScript changes on your web site, think about the following:

- An elaborate effect is cool the first time, but may become a nuisance when the same visitor comes back to your site to get some information they found the last time.
- Using animation in JavaScript (which has to be converted by a browser running inside an operating system) is incredibly hard to get smooth. Any other processing the computer has to do will interfere with your animations, and because there is no direct access to the video hardware (like OpenGL or ActiveX components have), there is no buffering to gloss over timing problems. In other words, if you use several animations that are dependent on each other, don't rely on timing to execute them. Instead, make sure that one animation initiates the next one once it has finished animating.
- Animation may pose an accessibility barrier as people with learning disabilities or epilepsy might not be able to cope with it. Make sure that if you use animation heavily, or you need animation that is very hectic, you also offer an option to turn it off.
- Don't rely on the animation to work when it comes to crucial elements of your site—like the menu. If an animation fails for some reason and the visitor cannot access your menu, you have lost them.
- While there are many examples where hiding and showing different page elements is beneficial to the usability of the page, don't get overly excited about this trick. First, the interface you created is not necessarily what the visitor sees (they might have JavaScript disabled, only see part of the screen in a magnifying tool, or have CSS turned off), and they'll have to download and deal with all the content (hiding something in CSS doesn't make it disappear from the document; it just hides it). Second, it can become very annoying to have to click and expand elements repeatedly to reach what you came for. This is especially annoying when the state is not stored and you have to do it all again on your next visit.
- Although it is easy to validate user data entry with JavaScript, it is not a foolproof method. If your only means of blocking out invalid data entry is JavaScript, all a malicious attacker needs to do is to turn it off. The other handy part for evildoers when it comes to JavaScript validation is that they can simply analyze your JavaScript and see what the rules are.

In essence, if you use animation and dynamic effects, be sure to use them in moderation and only when they benefit the user. There are not many things left you can do to amaze people on the Web with JavaScript bells and whistles. Advertisers and enthusiastic developers have done this for years already, and people got thoroughly fed up with waiting for animations to finish and trying to figure out how a menu works.

The two tasks

Let's now go through three examples of libraries with different approaches to the whole theme of what a library should do: jQuery by John Resig, MooTools by Valerio Proietti, and the Yahoo! User Interface Library (YUI).

The tasks we will try to achieve are

- Creating a hierarchical navigation
- Animating page elements

Creating a hierarchical navigation

We'll use the libraries to turn the following HTML list into a hierarchical and collapsible menu:

```
<ul id="nav">
  <li><a href="#">Item 1</a></li>
  <li><a href="#">Item 2</a>
    <ul>
      <li><a href="#">Item 1-1</a></li>
      <li><a href="#">Item 1-2</a></li>
      <li><a href="#">Item 1-3</a>
        <ul>
          <li><a href="#">Item 1-3-1</a></li>
          <li><a href="#">Item 1-3-2</a></li>
          <li><a href="#">Item 1-3-3</a></li>
          <li><a href="#">Item 1-3-4</a></li>
        </ul>
      </li>
      <li><a href="#">Item 1-4</a></li>
      <li><a href="#">Item 1-5</a></li>
      <li><a href="#">Item 1-6</a></li>
    </ul>
  </li>
  <li><a href="#">Item 3</a></li>
  <li><a href="#">Item 4</a>
    <ul>
      <li><a href="#">Item 4-1</a></li>
      <li><a href="#">Item 4-2</a></li>
      <li><a href="#">Item 4-3</a></li>
    </ul>
  </li>
  <li><a href="#">Item 5</a></li>
  <li><a href="#">Item 6</a></li>
</ul>
```

The idea is that a nested list structure like this is the semantically correct way to mark up (turn into HTML) a menu. The script should hide all the nested lists and add a CSS class to the li elements that contain a nested list. Furthermore, it should show and hide the nested lists when the visitor clicks the links in these parent list elements. All the other links should work as expected. When there is no JavaScript available, the list should not collapse at all, as shown in Figure 9-1.

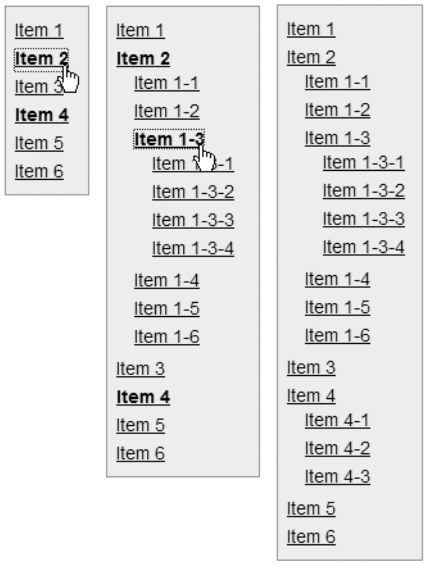


Figure 9-1. Turning a nested list into a hierarchical dynamic menu with JavaScript and how it looks without JavaScript

Animating page elements

One task JavaScript libraries are constantly used for is animation of page elements. We've covered this earlier in the chapter and will create a small example that slowly shows and hides a menu, as shown in Figure 9-2. You may remember that we advised against using animation in something crucial like a menu; however, this example will work without JavaScript because the link to show and hide the menu is generated via JavaScript and the DOM. When JavaScript is disabled, the show and hide link is not generated at all—the user does not get the promise of functionality that will not work. If you add the link in HTML and make it dependent on JavaScript to make sense, this wouldn't be the case.

A natural animation does not work in linear fashion, but either accelerates at the start and gets slower or starts slower and accelerates toward its end value (imagine a ball running out of momentum or a metal ball being dragged by a magnet). That is why some JavaScript libraries use Easing (`http://www.robertpenner.com/easing/`), which is a collection of premade animation algorithms that simulate these more natural movements. Not all libraries support this, but it is a nice way to make animations look more "real life."

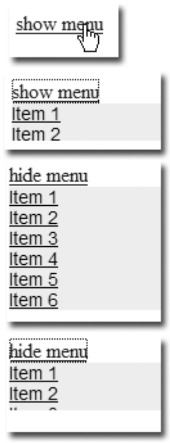


Figure 9-2. Animating the showing and hiding of a menu

Understanding and using jQuery

Probably one of the most aspiring libraries out there is John Resig's jQuery, available at `http://jquery.net`. The idea and goal of it becomes apparent in the introduction:

> *jQuery is a new type of JavaScript library. It is not a huge, bloated framework promising the best in Ajax—nor is it just a set of needlessly complex enhancements—**jQuery is designed to change the way that you write JavaScript**.*

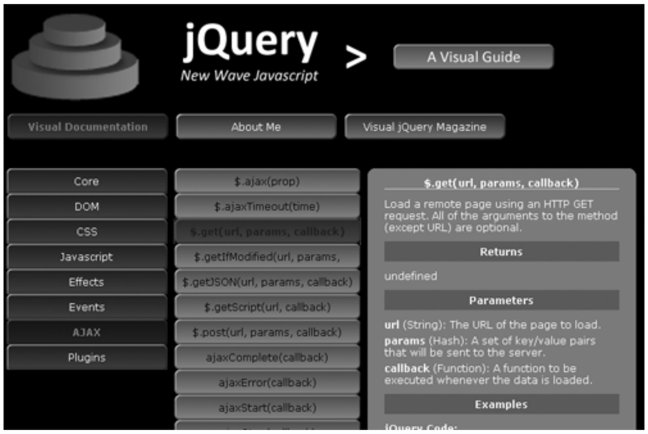
One of the main goals of jQuery is to keep the library itself and your code as small as possible. In order to achieve this it uses a technique called chainable methods, which means you can attach any of its functions to the other and create one logical stream of functionality. For example, the native JavaScript for calling a function called `example()` when any link in the document gets clicked is as follows:

```
var as = document.getElementsByTagName('a');
for (var i = 0; i < as.length; i++){
    as[i].onclick = example;
}
```

Using jQuery, this would be

```
$('a').click(example);
```

All the available functions of jQuery are explained and can be navigated at Visual jQuery (`http://www.visualjquery.com/index.xml`), as shown in Figure 9-3.



$.get(url, params, callback)

Load a remote page using an HTTP GET request. All of the arguments to the method (except URL) are optional.

Returns

undefined

Parameters

url (String): The URL of the page to load.

params (Hash): A set of key/value pairs that will be sent to the server.

callback (Function): A function to be executed whenever the data is loaded.

Examples

jQuery Code:

Figure 9-3. Visual jQuery is a visual representation of all the methods you have at your disposal with a quick explanation.

Hierarchical navigation in jQuery

The code to turn our demo list into a hierarchical navigation in jQuery looks like this. You can see that it is quite short, but also not that easy to understand if you don't adhere to a clean coding style with proper indentation. We won't go into details about the script, since there are many comments in this example explaining what each line does. Comments in JavaScript start with a double slash, //.

```
// when the document has finished loading
$( document ) . ready (
  function() {
    // add a class called "dynamic" to the element with the ID nav
    $( '#nav' ).addClass('dynamic');
     // loop through all UL elements inside the "nav" element
    $( '#nav ul' ).each(
      function(){
        // add a class called "parent" to the parent element
        // of the current UL
        $(this.parentNode).addClass('parent');
      }
    );
     // if the visitor clicks on any link inside the "nav" element
    $('#nav a').click(
      function(){
        // get all lists inside the parent node of this link
        var uls = this.parentNode.getElementsByTagName('ul');
        // if there is at least one
        if(uls.length>0){
          // check its style display attribute and toggle it from
         // block to none
          uls[0].display=uls[0].style.display=='block'?'none':'block';
          // don't follow the link
          return false;
        }
      }
    );
  }
);
```

Animation in jQuery

Animation of objects in jQuery is either done with the `animate()` method or with preset permutations of it, namely `fadeIn()`, `fadeOut()`, `fadeTo()`, `hide()`, `show()`, `slideDown()`, `slideUp()`, and `slideToggle()`, each of which do what their name says.

Most of these methods take parameters such as the speed of the animation, which can be defined either in milliseconds or as `slow`, `medium`, or `fast`. Some methods also need a final value of the element's property; for example, `fadeTo()` needs the speed of the animation and the final opacity as a value between 0 and 1 as parameters.

The animate() method itself has the handy option to define parameter values with text like hide, show, or toggle, which makes it easy to show and hide elements in a smooth fashion without having to know their initial measurements (something you have to do if you wanted to create the animation sequence with your own JavaScript methods).

```
// when the document has finished loading
$( document ) . ready (
  function() {
    // hide the navigation element
    $('#nav').hide();
    // create a link to show and hide the navigation
    $('#nav').before('<a href="#">show menu</a>');
    // define the states to execute alternately when the visitor
    // clicks the link
    $('#nav').prev().toggle(
      // set the text of the link to "hide menu" and show the menu
      function(){
        $(this).html('hide menu');
        $('#nav').slideToggle('medium');
      },
      // set the text of the link to "show menu" and hide the menu
      function(){
        $(this).html('show menu');
        $('#nav').slideToggle('medium');
      }
    );
  }
);
```

The animation capabilities of jQuery are easy to use and give you some shortcuts other libraries don't give you. On the other hand, at the time of this writing there is no option of changing the animation speed throughout the animation because jQuery does not feature Easing.

Understanding and using MooTools

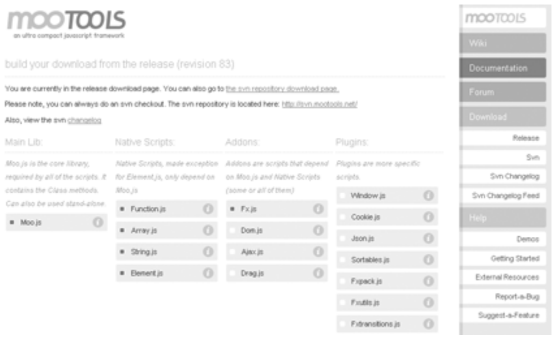
MooTools (http://MooTools.net/) is a library with a bit of a confusing history. Originally MooTools was a visual effect library called moo.fx (http://moofx.mad4milk.net/) that was built on top of what may be the biggest JavaScript framework, Prototype (http://prototype.conio.net/).

The developers soon realized that Prototype is a bit too much to use for small effects, reengineered moo.fx, and started to create their own base library to use rather than Prototype. Thus, MooTools was born. The core of MooTools and the main idea was to allow classical object-oriented programming like you'd use in other languages such as Java or C#. On top of that, the effect libraries of moo.fx have been incorporated; all this together makes up quite a nice library to use. In the words of the developers:

MooTools is a very compact, modular, object-oriented JavaScript framework. Its unique design makes it extremely cross browser, easy to use, and a snap to extend with your own code. It comes with a choice of more than fifteen scripts, plugins and add-ons, including Effects, based on (moo.fx) Ajax, based on (moo.ajax), Dom Navigator, based on (moo.dom), Drag and Drop, Sortable lists, cookies Manager and many more. All the previous moo scripts have been made better, reorganized and extended to fully take advantage of the new OO architecture. One of the big differences is The Element Extension: MooTools makes it possible for you to extend HTML elements with your own methods, to make your life easier and your coding style way cooler.

In other words, just like jQuery, MooTools disregards the current style of JavaScript and tries to improve and change the way we write scripts for the Web.

Undoubtedly the coolest thing about MooTools is the accompanying web site and especially the download section at `http://MooTools.net/download/release`, shown in Figure 9-4. The download section allows you to customize your version of MooTools to make sure you only get what you need instead of a really large library with lots of elements you'll never use.



build your download from the release (revision 83)

You are currently in the release download page. You can also go to the svn repository download page.

Please note, you can always do an svn checkout. The svn repository is located here: http://svn.mootools.net/

Also, view the svn changelog

Main Lib:	Native Scripts:	Addons:	Plugins:
Moo.js is the core library, required by all of the scripts. It contains the Class methods. Can also be used stand-alone.	*Native Scripts, made exception for Element.js, only depend on Moo.js*	*Addons are scripts that depend on Moo.js and Native Scripts (some or all of them)*	*Plugins are more specific scripts.*
■ Moo.js	■ Function.js	■ Fx.js	Window.js
	■ Array.js	Dom.js	Cookie.js
	■ String.js	Ajax.js	Json.js
	■ Element.js	Drag.js	Sortables.js
			Fxpack.js
			Fxutils.js
			Fxtransitions.js

Figure 9-4. The MooTools download section allows for a customizable download that only gives you what you need in either a readable format or optimized and packed for size. The interface also recognizes dependencies and every download will be a working chunk of the whole library.

MooTools also comes with extensive documentation (which you'll find at `http://docs.MooTools.net/`) listing all the options you have. The problem with the documentation is that there is no offline version available at the moment, which can be frustrating if you wanted to use MooTools on the go and you needed to look up something.

MooTools is a library that makes it appear easy to create dynamic interfaces that can be animated to fade in and out or slide around. The whole site is built with MooTools and is a case study for the things MooTools is capable of.

However, there are still some inconsistencies in the library that (we hope) will be fixed by the time you are reading this. The most obvious fault is that MooTools has no method to stop a link from being followed or a button from submitting a form. Most of the time you need this, though, as you make a link or a button call a JavaScript function instead of loading a new document or submitting a form. As best practices when it comes to scripting dictate that you don't rely on JavaScript, this makes it a real pain to develop to these practices. It is easy to create an inaccessible web application in MooTools, but if you want to enhance an already working web site that uses HTML and server-side scripting means you need to extend MooTools with a function that allows you to stop the default event from happening. There is a function like this in the code that follows soon.

Generally there is a lot of work going into MooTools, and many communication channels such as the MooTools forum (`http://forum.MooTools.net/`) are available where you can ask for help. The fresh look and feel of the site makes it interesting to dig around and find out more.

Hierarchical navigation in MooTools

MooTools approaches JavaScript much the same way jQuery does. Instead of using and extending existing DOM methods like `getElementsByTagName()` or `getElementById()`, it offers you shortcut methods like `$()` and allows for chaining of methods.

The example script of the hierarchical navigation therefore looks rather similar to the jQuery example, except for some differences in naming and syntax. Information about what each line does is provided as inline comments (the lines starting with //).

```
// when the DOM of the page is ready
Window.onDomReady(
  function(){
    // add the CSS class "dynamic" to the element with the ID "nav"
    $('nav').addClass('dynamic');
    // get all LI elements inside the element with the ID "nav"
    $lis = $('nav').getElements('li');
    // loop through all the list items
    $lis.each(
       // the each() method parses each list item as the parameter o
      function(o){
        // if the list item contains UL elements
        if($(o).getElements('ul').length>0){
          // add a CSS class to the list item with the name "parent"
          o.addClass('parent');
          // get the first link inside the list item
          var trigger = $E('a', o);
          // execute a function when the visitor clicks the link
          trigger.addEvent('click',
            function(e){
              // get the first nested UL element and toggle its display
              var nest = o.getElements('ul')[0];
              nest.style.display=nest.style.display==➥
'block'?'none':'block';
             // don't follow the link
              return Window.stopEvent(e);
            }
          )
        }
      }
    );
  }
);
// Hack extension to allow links not being followed when the user
// clicks them, this is not part of MooTools yet, but should be.
Window.extend({
  stopEvent: function(e){
    if (e.stopPropagation){
      e.stopPropagation();
      e.preventDefault();
    } else {
      e.returnValue = false;
      e.cancelBubble = true;
    } return false; }
});
```

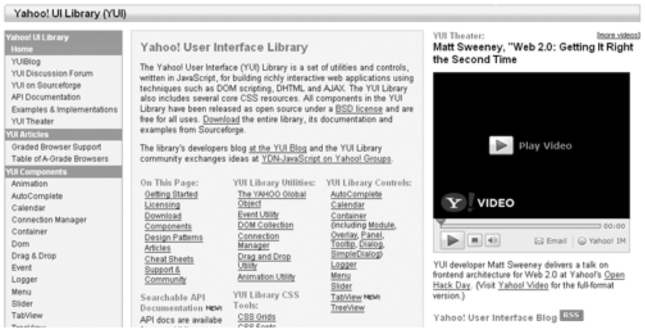
Animation in MooTools

With moo.fx as its ancestor, MooTools comes with an impressive set of predefined animation sequences and effects. You can create effects with the Fx object, and each effect has different methods to control it such as show(), hide(), or toggle().

```
// when the DOM of the page is ready
Window.onDomReady(
  function(){
    // add the CSS class "dynamic" to the element with the ID "nav"
    // thus hiding it
    $('nav').addClass('dynamic');
    // create a new link element, set the href attribute to # (to make
    // it appear as a link) and add text inside the link
    // saying "show menu"
    var trigger = new Element('a');
    trigger.setProperty('href','#');
    trigger.appendText('show menu');
    // add the new link before the menu to the document
    $(trigger).injectBefore('nav');
    // define a new slide effect for the menu
    var slideeffect = new Fx.Slide('nav');
    // hide the menu initially
    slideeffect.hide();
    // make the newly added link execute a function when a visitor
    // clicks it
    $(trigger).addEvent(
    'click',
      // function to toggle the menu state
      function(e){
        // alternately show and hide the menu and slide it open or
        // closed
        slideeffect.toggle('vertical');
        // change the text of the trigger link accordingly
        if($(trigger).innerHTML === 'show menu'){
          $(trigger).innerHTML = 'hide menu';
        } else {
          $(trigger).innerHTML = 'show menu';
        }
        // don't follow the link
        return Window.stopEvent(e);
      }
    )
  }
);
// Hack extension to allow links not being followed when the user
// clicks them
Window.extend({
  stopEvent: function(e){
    if (e.stopPropagation){
      e.stopPropagation();
      e.preventDefault();
    } else {
      e.returnValue = false;
      e.cancelBubble = true;
    } return false; }
});
```

Understanding and using YUI

The Yahoo! User Interface Library (YUI), which you'll find at http://developer.yahoo.com/yui, was developed primarily to ease the development process of web sites inside Yahoo!. Instead of each development team consistently having to face and fix browser issues and create scripts that do the same things over and over again, YUI (shown in Figure 9-5) acts as a central repository for these tasks. Developers feed problems they encounter back to the YUI team and those problems get fixed in the next version of the library.



Yahoo! User Interface Library

The Yahoo! User Interface (YUI) Library is a set of utilities and controls, written in JavaScript, for building richly interactive web applications using techniques such as DOM scripting, DHTML and AJAX. The YUI Library also includes several core CSS resources. All components in the YUI Library have been released as open source under a BSD license and are free for all uses. Download the entire library, its documentation and examples from Sourceforge.

The library's developers blog at the YUI Blog and the YUI Library community exchanges ideas at YDN-JavaScript on Yahoo! Groups.

API docs are availabe

Matt Sweeney, "Web 2.0: Getting It Right the Second Time

YUI developer Matt Sweeney delivers a talk on frontend architecture for Web 2.0 at Yahoo!'s Open Hack Day. (Visit Yahoo! Video for the full-format version.)

Figure 9-5. The Yahoo! User Interface Library web site

The idea behind the different YUI components is that they help developers with one issue at a time rather than offering an all-encompassing approach the way that jQuery or MooTools does with methods like `$()`. To this end, YUI is separated into several components, each of which is designed to help with one part of web scripting:

- Animation—Allows you to animate parts of a document, cross-browser and securely
- Connection—Allows you to connect to the server to retrieve data via Ajax
- Dom—Allows you to retrieve, alter, and add elements from and to the document
- DragDrop—Allows you to create drag-and-drop interfaces
- Event—Allows you to get informed and react to happenings in the browser and to the document (for example, a user clicking a link)
- Fonts/Grids/Reset—CSS components that allow you to create layouts that work consistently across browsers

The library is separated into library components and widgets. While the components mentioned earlier allow you to easily create your own scripts, widgets are interface elements that can be used as they are and customized to your needs. At the time of this writing, YUI has the following widgets:

- Logger—Allows you to debug an application while developing it independent of the browser
- Menu/TreeView—Allows you to create menus for your site or application ranging from simple static menus via dynamic hierarchical menus up to contextual menus
- Slider—Provides you with a method to create slider controls or color pickers
- Tabs—Allows you to develop tabbed interfaces
- Autocomplete—Allows you to create text boxes that offer possible values to go in them while you are typing
- Calendar—Provides you with an out-of-the-box calendar widget to make date picking in a form a lot easier for the visitor
- Container—Allows for overlays, tooltips, modular windows, and dialog boxes

YUI may have the most extensive documentation and set of code examples of all the libraries. However, the examples can be a bit exhaustive as they try to show every option a component or widget has rather than explain step by step what the options are. A couple of really good companions if you want to use YUI are the Cheatsheets (which are available as PDFs) and the mailing list, which you can use to find help and search for instances where your problem has already been solved.

Because YUI code is meant to work in large web sites and applications, its main focus is to be stable across all modern browsers (what modern browsers are and how Yahoo! defines support is explained in the browser support grid at `http://developer.yahoo.com/yui/articles/gbs/gbs.html`) and safe to implement together with other scripts. The latter is achieved by adding namespaces to the code. This means that every function developed using YUI has to start with the word `YAHOO` followed by what it is and its name. This might appear verbose at first sight, but it ensures that no Yahoo! script overwrites other functionality. Here are some examples:

- `YAHOO.example.createDynamicMenu()` is an example script using YUI that will create a dynamic menu.
- `YAHOO.util.Event` is the event utility component.
- `YAHOO.widget.Calendar` is YUI's calendar widget.

Following this convention and sticking to a standard JavaScript syntax rather than allowing for shortcuts like jQuery and MooTools does mean that scripts using YUI tend to be larger. What it also means, though, is that the scripts don't need to be converted back to something the JavaScript parser understands.

Hierarchical navigation using YUI

The easiest option for creating the example of a collapsible hierarchical navigation is to use either the Menu (`http://developer.yahoo.com/yui/menu/`) or the TreeView widget (`http://developer.yahoo.com/yui/treeview/`). However, to demonstrate how to use the different components of the library in your own script, let's give it a go. Notice once again that the explanations of what the script does are in the inline comments starting with a double slash.

```
// use the example namespace and call the main object "tree"
YAHOO.example.tree = {

  // the init function to hide elements and add the "parent" classes
  init:function(){

    // add a class called dynamic to the element that was sent as a
    // parameter (stored in this)
    YAHOO.util.Dom.addClass(this,'dynamic');
    // add an event listener that calls the toggle method when a
    // visitor clicks anywhere inside the tree element
    YAHOO.util.Event.addListener(this, 'click', ➥
YAHOO.example.tree.toggle);

    // grab all UL elements inside the element and loop through
    // each of them
    var uls = this.getElementsByTagName('ul');
    for(var i=uls.length-1;i>-1;i--){

      // add a class called "parent" to each LI that contains a UL
      YAHOO.util.Dom.addClass(uls[i].parentNode, 'parent');

    }
  },

  // the method to show and hide the nested lists
  toggle:function(e){

    // use the Event component to find out which element was clicked on
    var t = YAHOO.util.Event.getTarget(e);

    // compare if the element had the name "a", thus being a link
    if(t.nodeName.toLowerCase()==='a'){

      // check if the link is inside an LI that has a nested UL element
      var nestedLists = t.parentNode.getElementsByTagName('ul');
      if( nestedLists.length > 0 ){

        // toggle the display of the nested UL
        nestedLists[0].style.display = nestedLists[0].style.➥
display === 'block'?'none':'block';

        // don't follow the link
        YAHOO.util.Event.preventDefault(e);
      }
    }
  }
}

// as soon as the element with the ID 'nav' is available, call the
// init method
YAHOO.util.Event.onAvailable('nav', YAHOO.example.tree.init);
```

The structure of a script using YUI does not differ much from any other traditional JavaScript, which makes it look a bit cumbersome compared to the slicker “new” way of scripting that jQuery or MooTools offer. However, the practical upshot of this is that you don’t need to learn about the library itself to use it. In a professional or distributed development environment, this can be a very important asset, as you normally don't have the time and budget to train people on systems that should make their job easier.

Animation using YUI

Animation is a component of YUI that uses the aforementioned Easing methods to allow you to create natural-looking animations. Unlike the other libraries we've mentioned, YUI does not provide premade methods to show and hide or toggle elements; you are expected to create your own instead. This seems a less practical approach, but on the other hand you have much more control over the outcome. There are lots of examples on the homepage (`http://developer.yahoo.com/yui/animation/`) and inside the YUI download zip to get you going. The overall structure of creating an animation is pretty straightforward. You define an animation with this constructor:

```
   var anim = new YAHOO.util.Anim(element, { properties }, duration, ➥
Easing method );
```

- The properties define how you want to animate the element; for example, `width: {from:100, to: 500}`.
- The duration is the length of the animation in seconds.
- The easing method specifies which one of the preset methods for natural animation you want to use (for example, `easeIn`).

You start the animation by calling the method `animate()`, and you could stop it prematurely with the `stop()` method. You can also execute different functions before, during, and after the animation using the `onStart`, `onTween`, and `onComplete` custom events, respectively.

Custom events are a specialty of YUI. Normal JavaScript events are interesting moments in the life cycle of a web page. Examples are when the page was loaded (the load event), when a user clicks an element inside the page (the click event), or when the browser window is resized (the resize event). You can listen for these events and do something when they occur. YUI also allows you to define your own events for the current document that are not part of the normal browser toolset. You can define functions that get executed when these events are fired (listeners) and make the event happen any time you want to. This is extremely handy if you have a complex application or you want to make sure that several page elements change when something happens to another one. If you go to Yahoo! Maps in Europe (`http://uk.local.yahoo.com/maps`*) you can see custom events in action; we used about 20 different ones to make this interface work smoothly.*

Putting all of this together, we once again have a script that does look a lot bigger than the jQuery or MooTools equivalents, but also makes it easy to debug and find your way even if you don't know YUI. Explanations are once again in comments (the lines starting with `//`).

```
YAHOO.example.animateMenu = {
  // predefine a property to store the visibility status of the menu
  menuvisible:false,
  // initialization method
  init:function(){
    // store the height of the menu in menuheight
    YAHOO.example.animateMenu.menuheight = this.offsetHeight;
    // add a CSS class called dynamic to the menu, thus hiding it
    YAHOO.util.Dom.addClass(this, 'dynamic');
    // create a new link element and give it a href to make it display
    // as a link
    YAHOO.example.animateMenu.trigger = document.createElement('a');
    YAHOO.example.animateMenu.trigger.setAttribute('href', '#');
    // add the text "show menu" to the link
    YAHOO.example.animateMenu.trigger. ➥
appendChild(document.createTextNode('show menu'));
    // insert the link before the menu
    this.parentNode.insertBefore(YAHOO.example.animateMenu.trigger, ➥
this);
    // call the method togglemenu when a user clicks the link
    YAHOO.util.Event.addListener(YAHOO.example.animateMenu.trigger, ➥
'click',
    YAHOO.example.animateMenu.togglemenu );
  },
  // method to show and hide the menu
  togglemenu:function(e){
    // if the menu is hidden
    if(YAHOO.example.animateMenu.menuvisible === false){
    // set the menu height to zero (to avoid it flashing up before the
    // animation starts)
      YAHOO.util.Dom.setStyle('nav', 'height', '0');
      // remove the CSS class to show the menu
      YAHOO.util.Dom.removeClass('nav', 'dynamic');
      // define the end value of the animation as the original height
      // of the menu
      var end = YAHOO.example.animateMenu.menuheight;
      // animation attributes - animate until the height is the
      // original height
      var attributes = {height:{to:end}};
      // set the property to state that the menu is visible
      YAHOO.example.animateMenu.menuvisible = true;
      // change the text of the link to "hide menu"
      var linktext = 'hide menu';
    // if the menu is visible...
    } else {
      // define the end value of the animation as 0
      var attributes = {height:{to:0}};
      // set the property to state that the menu is hidden
      YAHOO.example.animateMenu.menuvisible = false;
```

```
          // change the text of the link to "show menu"
          var linktext = 'show menu';
        }
        // define a new animation to change the element with the ID nav
        // using the defined attributes, a duration of one second
        // and start and end slower using the Easing methods.
        var anim = new YAHOO.util.Anim('nav', attributes, 1, ➥
    YAHOO.util.Easing.easeBoth);
        // start the animation
        anim.animate();
        // when the animation is finished
        anim.onComplete.subscribe(
          function(){
            // change the text of the link
            YAHOO.example.animateMenu.trigger.firstChild.nodeValue = ➥
    linktext;
          }
        );
        // don't follow the original link
        YAHOO.util.Event.preventDefault(e);
      }
    }
    // execute the initialization method as soon as the element
    // with the ID "nav" is available.
    YAHOO.util.Event.onAvailable('nav', YAHOO.example.animateMenu.init);
```

The animation component of YUI is massive in terms of features and functionality and allows for a lot of different effects and ideas. It does mean, however, that you need to write these effects yourself based on a set of tools. This is harder to do than, say, using jQuery's `show('fast')`, but it also gives you a lot more options.

Summary

We hope this chapter has given you some insight into the process and the concept behind using special effects in web sites and blogs. We've deliberately tried to keep the examples to a bare minimum and didn't delve too much into details about the various libraries. After all, this is not a JavaScript book (there are other books to teach you the basics of this language), and the JavaScript library environment is such a fast-moving target that any book written on it will be outdated in a month's time. By the time this one comes out, some of the things explained even in these short introductions might have changed.

Our intention was to make sure that if you are going to use special effects on your site, you do so for a reason and you understand the implications of it. Far too many web sites out there go heavy on JavaScript for the sake of using it or to try to draw visitors' attention away from lack of content or updates. This only works for a very short period of time and is nothing to build your network of contacts or visitors on.