AdvancED DOM Scripting

Dynamic Web Design Techniques

Jeffrey Sambells
with Aaron Gustafson

friendsof ED™
DESIGNER TO DESIGNER™
an Apress® company

AdvancED DOM Scripting: Dynamic Web Design Techniques

ISBN-13 (pbk): 978-1-59059-856-6

ISBN-10 (pbk): 1-59059-856-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Credits

Lead Editor
Chris Mills

Technical Reviewers
Cameron Turner, Victor Sumner

Editorial Board
Steve Anglin, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jason Gilmore, Jonathan Hassell, Chris Mills, Matthew Moodie, Jeffrey Pepper, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Senior Project Manager
Kylie Johnston

Copy Edit Manager
Nicole Flores

Copy Editor
Heather Lang

Assistant Production Director
Kari Brooks-Copony

Senior Production Editor
Laura Cheu

Compositor
Dina Quan

Artist
Kinetic Publishing Services, LLC

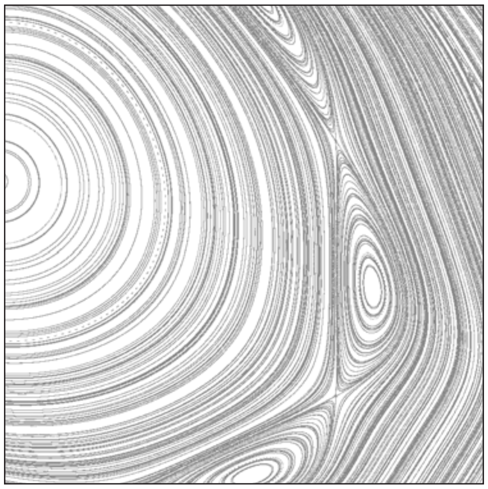
Proofreader
Liz Welch

Indexer
Broccoli Information Management

Cover Image Designer
Bruce Tang

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski



CHAPTER 5

DYNAMICALLY MODIFYING STYLE AND CASCADING STYLE SHEETS

In Chapter 1, I stressed the importance of separating your markup, behavior, and presentation. The problem with separating everything is that many aspects of the three layers overlap or even collide in an explosion messed-up code. You've already seen how to manipulate the DOM document structure in Chapter 3 and the behavioral event listeners in Chapter 4. Now it's time to discuss the presentation layer and how you can write scripts to modify your presentation. We'll explore ways to modify the presentation without embedding it in the script itself, though sometimes doing so will be unavoidable.

In this chapter, we'll briefly look at the key parts of the W3C DOM Level 2 Style specification along with a variety of methods you can use to keep your presentation separated from your DOM scripts. We'll also discuss one of the biggest misconceptions about the `style` property as well as a little fix for PNG transparency in some versions of Microsoft Internet Explorer. The last part of the chapter will also deal with creating transitions and effects, which will be further expanded on in Chapter 10.

The W3C DOM2 Style specification

The W3C DOM2 Style specification (`http://w3.org/TR/DOM-Level-2-Style/`) can appear very scary at first, but if you understand the basics of CSS, you already know the majority of it. Without discussing CSS at length, I'm going to focus on a few of the more complicated, and more useful, aspects of the specification, focusing on things you'll encounter in your day-to-day DOM scripts.

CSSStyleSheet objects

The `CSSStyleSheet` objects represent all of your CSS. These include external and inline style sheets specified using `<style type="text/css"></style>` tags. `CSSStyleSheet` is constructed from various other DOM2 CSS objects, such as a list of `CSSStyleRule` objects representing each of the rules in the style sheet.

You'll find a list of your document's `CSSStyleSheet` objects in the `document.styleSheets` property. Each of the `CSSStyleSheet` objects has several properties:

- `type` will always be `text/css`, as you're dealing with Cascading Style Sheets in your HTML documents.
- `disabled` is a Boolean property indicating that the style sheet is applied (`false`) to the current document or disabled (`true`).
- `href` is the URL where the style sheet is located relative to the current document, or it's the URL of the current document for inline style sheets.
- `title` is a label that can be used to group style sheets, as you'll see later in this chapter.
- `media` indicates the type of target devices the style applies to, such as `screen` or `print`.
- `ownerRule` is a read-only `CSSRule` object representing the parent rule if the style sheet was imported using `@import` or something similar.
- `cssRules` is a read-only `CSSRuleList` list of all the `CSSRule` objects in the style sheet.

Each `CSSStyleSheet` object also has methods, including the following:

- `insertRule(rule,index)`: For adding new style declarations
- `deleteRule(index)`: For removing rules from the style sheet

We'll look at these properties and methods in more detail throughout this chapter as you modify and edit the presentation of your documents.

CSSStyleRule objects

Within each `CSSStyleSheet` is a list of `CSSStyleRule` objects. These objects each represent a single rule such as

```
body {
    font: 62.5%/1.2em "Lucida Grande", Lucida, Verdana, sans-serif;
    background: #c7f28e;
    color: #1a3800;
}
```

The `CSSStyleRule` objects have properties that define the rule:

- type is a property inherited from the CSSRule object and is a number between 0 and 6 representing the type of rule. For CSSStyleRule rules this will always be 1, but other rule objects, such as CSSImportRule, which represents @import rules, take other values. The list of type values follows:
  - **0**: CSSRule.UNKNOWN_RULE
  - **1**: CSSRule.STYLE_RULE (indicates a CSSStyleRule object)
  - **2**: CSSRule.CHARSET_RULE (indicates a CSSCharsetRule object)
  - **3**: CSSRule.IMPORT_RULE (indicates a CSSImportRuleobject)
  - **4**: CSSRule.MEDIA_RULE (indicates a CSSMediaRule object)
  - **5**: CSSRule.FONT_FACE_RULE (indicates a CSSFontFaceRule object)
  - **6**: CSSRule.PAGE_RULE (indicates a CSSPageRule object)
- cssText contains the string representation of the entire rule in its current state. If the rule has been modified using other DOM methods, this string will represent the changes as well.
- parentStyleSheet references the parent CSSStyleSheet object.
- parentRule will reference another CSSRule if the rule is within another rule. For example, rules within a specific @media rule will have the @media rule as their parent rule.

Also, the CSSStyleRule objects have CSS-specific properties, such as the following:

- selectorText contains the selector used for the rule.
- style, like HTMLElement.style, is an instance of CSSStyleDeclaration.

You'll also see these properties and methods in more detail throughout this chapter.

CSSStyleDeclaration

The last object we'll look at, and probably the one you'll run into most, is the CSSStyleDeclaration object, which is the object that represents an element's style property. Like the CSSStyleRule object, the CSSStyleDeclaration object has properties such as the following:

- cssText contains the string representation of the entire rule.
- parentRule will reference the CSSStyleRule.

Also, the CSSStyleDeclaration object has several methods, including these:

- getPropertyValue(propertyName) returns the value of a CSS style property as a string.
- removeProperty(propertyName) removes the given property from the declaration.
- setProperty(propertyName,value,priority) allows you to set the value of a given CSS property.

Instances of the CSSStyleDeclaration object, such as the HTMLElement.style property, also have a shortcut access method through CSS2Properties, but we'll discuss that in more detail when we look at the style property later in this chapter.

As you'll see later in this chapter, the `style` *property doesn't reference the computed style from the cascade.*

A lack of support

Unfortunately, some browsers don't support all the features of the Style specification. Those browsers have similar objects that represent similar features, but they're accessed differently than the DOM2 Style methods. Throughout the rest of this chapter, I'll focus on the W3C methods, but all of the style-related access and manipulation methods you'll be adding to your ADS library will be browser agnostic and include any necessary browser-specific access methods. I'll point out the cases where access differs from the W3C, but I won't go into a lot of detail regarding the proprietary methods. The only exception will be the Microsoft Internet Explorer CSS `filter` property in the context of translucent PNGs, which I'll cover near the end of this chapter.

When DOM scripting and style collide

The presentation layer in your web application consists primarily of the CSS styles. I'm going to assume that you're at least somewhat familiar with CSS and that you already know the basics, but if you're not, or haven't yet embraced CSS, I highly recommend these additional books:

- *Web Standards Solutions: The Markup and Style Handbook* by Dan Cederholm (friends of ED; ISBN-13: 978-1-59059-381-3)
- *CSS Mastery: Advanced Web Standards Solutions* by Andy Budd with Simon Collison and Cameron Moll (friends of ED; ISBN-13: 978-1-59059-614-2)
- *Transcending CSS: The Fine Art of Web Design* by Andy Clark and Molly Holzschlag (New Riders Press; ISBN-13: 978-0-32141-097-9)

CSS, like DOM scripting, is not a magic fix for your web site. To take advantage of its power and flexibility, you need to implement it properly with all your CSS styles defined in an external style sheet. This will separate the style from the markup, but it poses two problems:

- As I said, CSS is not a magic solution. It often requires additional tags in the semantic markup so that the CSS can manipulate extra elements for stylistic purposes.
- CSS also poses a problem for your behavioral layer. In many, if not most, cases, your behavioral enhancements will involve manipulating the presentation of your elements—but your presentation shouldn't be mixed into your DOM scripts either.

With a bit of forethought and planning, the presentation of your document can be separated from your behavioral elements while maintaining flexibility for both the CSS and DOM script.

Modifying markup for style

Semantic purists will discourage the use of any markup that doesn't directly relate to the semantics of the document, but sometimes it can't be avoided. As you work with CSS, you'll soon realize that its limitations will sometimes require a little extra markup to nudge things around and make them look the way you want—for presentation purposes only—but that's not a bad thing. Personally, I believe a little bit of extra markup isn't going to hurt anyone as long as it's done with forethought and you're not just layering on new markup without considering alternative solutions.

CSS image replacement is a common situation where additional markup is often required. The image could be replacing something as simple as the text in a header or something more complicated, such as a bulleted list or table. Either way, the goal is to provide a visually pleasing alternative to the text, but it should also maintain accessibility to the information.

The resulting markup should still

- be accessible to screen readers
- be understandable when images are off and CSS is on (a problem for many solutions)
- maintain accessibility features such as alt tags and titles
- avoid unnecessary markup—if at all possible

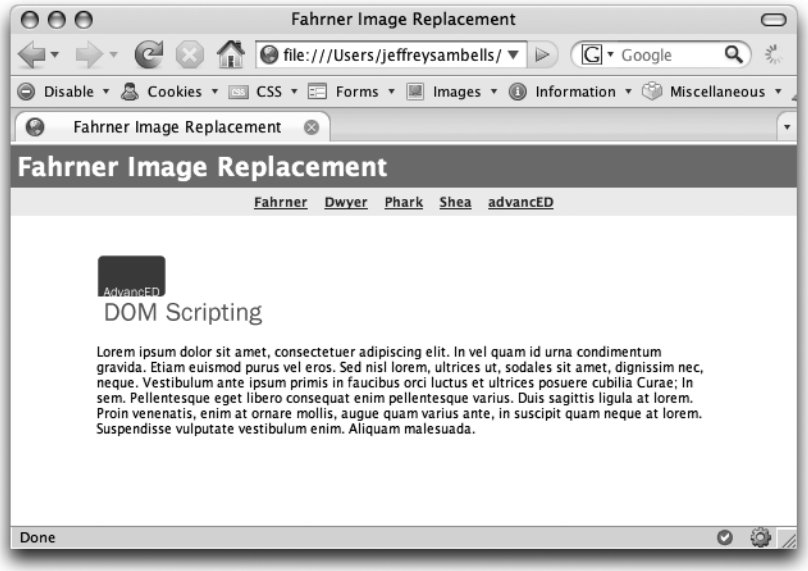
The classic technique is known as Fahrner Image Replacement (FIR). This technique involves adding an extraneous `<span>` tag around the content within an element such as a header:

```
<h1 id="advancedHeader"><span>Advanced DOM Scripting</span></h1>
```

and then applying CSS to insert a background image on the header and hide the `<span>`:

```
/* Add a background image to the parent element */
#advancedHeader {
    height:60px;
    background: transparent url(http://advanceddomscripting.com/➥
images/advancED-replace.png) no-repeat;
}
/* Hide the text */
#advancedHeader span {
    display:none;
}
```

The result, with the CSS applied, is a custom styled graphic that replaces the plain browser text, as shown in the `chapter5/image-replacement/fir.html` file in Figure 5-1; the plain text AdvancED DOM Scripting header has been replaced with an image.



Fahrner Image Replacement

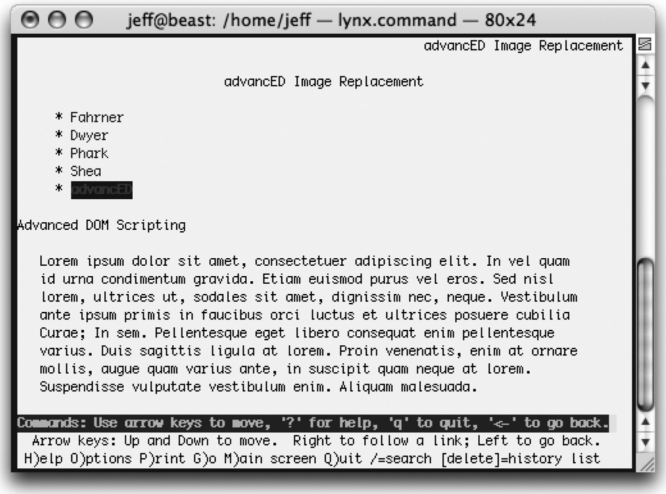
Fahrner Dwyer Phark Shea advancED

AdvancED
DOM Scripting

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In vel quam id urna condimentum gravida. Etiam euismod purus vel eros. Sed nisl lorem, ultrices ut, sodales sit amet, dignissim nec, neque. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; In sem. Pellentesque eget libero consequat enim pellentesque varius. Duis sagittis ligula at lorem. Proin venenatis, enim at ornare mollis, augue quam varius ante, in suscipit quam neque at lorem. Suspendisse vulputate vestibulum enim. Aliquam malesuada.

Figure 5-1. The header replaced with an image

Without CSS, the header will gracefully degrade into the plain text alternative shown in Figure 5-2.



```
                                                 advancED Image Replacement

                          advancED Image Replacement

     * Fahrner
     * Dwyer
     * Phark
     * Shea
     * advancED

Advanced DOM Scripting

   Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In vel quam
   id urna condimentum gravida. Etiam euismod purus vel eros. Sed nisl
   lorem, ultrices ut, sodales sit amet, dignissim nec, neque. Vestibulum
   ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia
   Curae; In sem. Pellentesque eget libero consequat enim pellentesque
   varius. Duis sagittis ligula at lorem. Proin venenatis, enim at ornare
   mollis, augue quam varius ante, in suscipit quam neque at lorem.
   Suspendisse vulputate vestibulum enim. Aliquam malesuada.

Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Figure 5-2. The header as it would appear in the text-based lynx browser

There are two problems with the classic FIR method. First, nothing shows if images are disabled, because the span is still hidden. The second problem is that `display:none` may also hide the content from screen readers that support CSS, effectively destroying the whole point of accessible image replacement.

Other methods have since been devised, such as the following three:

The Dwyer method by Leon Dwyer uses a 0 size for the extra <span>, but like FIR, it doesn't work with CSS turned on and images off, as in the chapter5/image-replacement/dwyer.html example:

```
/* Add a background image to the parent element */
#advancedHeader {
    height:60px;
    background: transparent url(http://advanceddomscripting.com/➥
images/advancED-replace.png) no-repeat;
}
/* Hide the text using a 0-sized box*/
#advancedHeader span {
    display:block;
    width:0;
    height:0;
    overflow:hidden;
}
```

The Phark method by Mike Rundle doesn't require any additional markup and uses a negative text indent to hide the content, but again, it doesn't work with CSS on and images off, as in the chapter5/image-replacement/phark.html example:

```
/* Use a background image along with a negative
text indent to hide the text */
#advancedHeader {
    text-indent: -100em;
    overflow:hidden;
    height:60px;
    background: transparent url(http://advanceddomscripting.com/➥
images/advancED-replace.png) no-repeat;
}
```

Another option is the Shea method by Dave Shea, which still uses an extra empty <span> element:

```
<h1 id="advancedHeader" title="AdvancED DOM Scripting">
    <span></span>Advanced DOM Scripting
</h1>
```

but rather than hiding the text, the span is positioned above the text with a solid opacity to hide the underlying words, as in the chapter5/image-replacement/shea.html example:

```
/* Use relative positioning for the parent
   so children can use absolute */
#advancedHeader {
    height:60px;
    position:relative;
}
/* Hide the text using an opaque background image
   on the absolute positioned span */
#advancedHeader span {
    background: transparent url(http://advanceddomscripting.com/➥
images/advancED-replace.png) no-repeat;
    position:absolute;
    display:block;
    width:100%;
    height:100%;
}
```

When images are off, the Shea method allows the text to remain visible. Also, the title attribute of the header can still act as the equivalent alt attribute on the image when hovering the pointer over the header.

Removing the extra markup

The extra <span> tag in the preceding image-replacement solutions provides no additional semantic meaning and is really an extra tag. It's not all that bad, but you could use a DOM script to achieve the same goal. If you're happy letting JavaScript-disabled browsers receive the degraded text-only version, you can easily add this extra markup by using a load event listener. To do so, start with a clean header that contains only the necessary identifying markup, as found in the chapter5/image-replacement/advancED.html source:

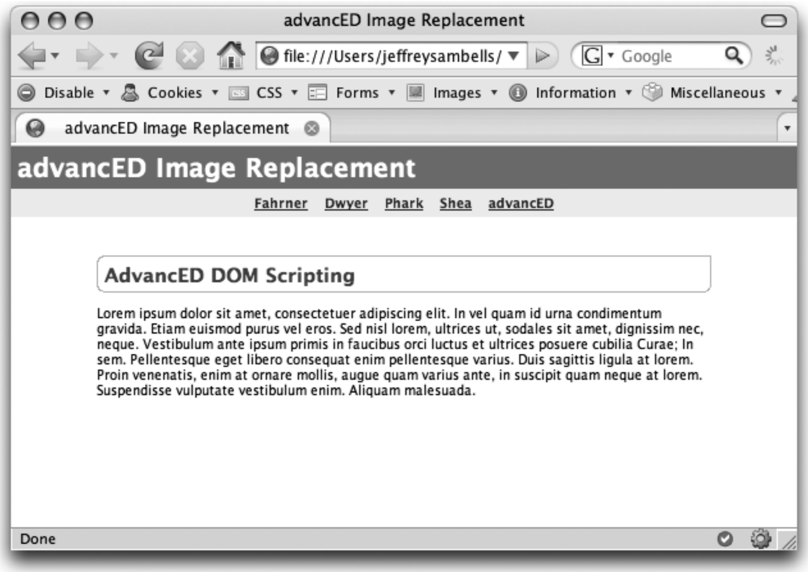
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>advancED Image Replacement</title>
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="../chapter.css" />

    <!-- ADS Library (full version from source linked here) -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>

    <!-- Image replacement -->
    <link rel="stylesheet" type="text/css"
        href="advancED.css" media="screen">

    <!-- The load script -->
    <script type="text/javascript" src="advancED.js"></script>

</head>
<body>
    <h1>advancED Image Replacement</h1>
    <h2 id="advancedHeader">Advanced DOM Scripting</h2>
</body>
</html>
```

Next, include the appropriate CSS rules in the linked advancED.css file. Here, I've chosen to use the Shea method as the base CSS method, but I've added an additional .advancED class to the selectors:

```
/* The styling for the text if images are disabled */
#advancedHeader {
    color: #1A5B9D;
}

/* Sizing the header for the image */
#advancedHeader.advancED {
    height:60px;
    position:relative;
    overflow:hidden;
}
/* Hide the text with the opaque image*/
#advancedHeader.advancED span {
    background: white url(http://advanceddomscripting.com/➥
images/advancED-replace.png) no-repeat;
    display:block;
    width:100%;
    height:100%;
    position:absolute;

}
```

The extra `.advancED` class is the tricky part, as it doesn't yet appear anywhere in the `chapter5/image-replacement/advancED.html` source markup.

With these CSS rules in place as is, a device running without JavaScript will see the header with the limited CSS (or no CSS if that's not available either), as shown in Figure 5-3.


advancED Image Replacement
file:///Users/jeffreysambells/
Google
Disable
Cookies
CSS
Forms
Images
Information
Miscellaneous
advancED Image Replacement
Done

advancED Image Replacement

Fahrner Dwyer Phark Shea advancED

AdvancED DOM Scripting

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In vel quam id urna condimentum gravida. Etiam euismod purus vel eros. Sed nisl lorem, ultrices ut, sodales sit amet, dignissim nec, neque. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; In sem. Pellentesque eget libero consequat enim pellentesque varius. Duis sagittis ligula at lorem. Proin venenatis, enim at ornare mollis, augue quam varius ante, in suscipit quam neque at lorem. Suspendisse vulputate vestibulum enim. Aliquam malesuada.

Figure 5-3. The advanced image replacement solution when the script hasn't run

To make everything really snazzy, the load event in the chapter5/image-replacement/advancED.js file is used to inject the extra <span> into the header and add the extra .advancED class name, which completes the transformation:

```
ADS.addEvent(window, 'load', function() {
    // Retrieve the header
    var header = ADS.$('advancedHeader');
    // Create an image element
    var image = document.createElement('IMG');

    // Only add the span and class if the image loads
    ADS.addEvent(image, 'load', function() {

        var s = document.createElement('SPAN');
        // Prepend the span to the header's children
        ADS.prependChild(header,s);

        // Create the title attribute as necessary
        if(!header.getAttribute('title')) {
            var i, child;
            var title = '';
            // Loop through the children and assemble the title
            for(i=0 ; child = header.childNodes[i] ; i++ ) {
                if(child.nodeValue) title += child.nodeValue;
            }
            header.setAttribute('title',title);
        }
        // Modify the class name to indicate the
        // change and apply the CSS
        header.className = 'advancED';
    });

    // Load the image
    // This hardcoded path isn't ideal.
    // You'll revisit this later in the chapter
    image.src = 'http://advanceddomscripting.com/➥
images/advancED-replace.png';

});
```

This example isn't very portable. You must change the script if you want to change the image. We'll solve this problem later in this chapter.

If the image loads successfully, the image's load event will generate the proper markup and trigger the CSS to do its thing:

```
<h1 id="advancedHeader" title="Advanced DOM Scripting" class="advancED">
    <span></span>
Advanced DOM Scripting</h1>
```

The addition of the class is important for degradation and cleanliness. By appending the class with your DOM script, you indicate to the CSS presentation layer that the change to the markup was successful and that the header should be modified. This allows you to keep the actual presentation separate from the DOM script while allowing the DOM script to control the process. At the same time, if either CSS or images are unavailable, the gracefully degraded version will still contain the header text as it appears in the markup.

If you try this example as is, you may run into problems when you override an existing class on the header elements. You'll be creating class name manipulation methods for your ADS library to solve that problem later in this chapter.

The only caveat with this method is that the replacement relies on your DOM script. Depending on the speed of the computer and the Internet connection, there may be a brief flicker when the browser renders the text and the load modification runs (while the image is loading). You'll have to weigh the pro of a clean, accessible, and degradable solution against the con of requiring JavaScript and decide which is right for you.

Keeping style out of your DOM script

Separating your style from your markup is easily done with the appropriate class and ID attributes along with an external CSS style sheet. But, the same concepts are often overlooked in your DOM scripts where it's equally important to separate stylistic changes from behavioral changes. To work properly, your DOM scripts will influence the appearance of your document—but they shouldn't do so in a way that forces you to edit the DOM script when tweaking the design.

In the previous section, you saw how image replacement could be achieved using a combination of CSS and DOM scripting to manipulate the markup. In this section, you'll see that there are a number of ways to influence appearance without directly changing stylistic properties. We'll also start off by taking a look at one of the most misunderstood things in DOM scripting: the DOM `style` property.

The style property

When applying presentation changes on a small scale, for maybe only one or two elements at a time, you've probably come across a number of examples online—and even in this book—that use the `style` property of the `HTMLElement` to modify its presentation. There are two big problems with modifying the `style` property:

- Using the `style` property embeds stylistic design into your behavioral DOM script, which isn't any better than using the `style` attribute in the semantic markup. Sometimes, however, using the `style` property does make sense for behavior-related properties, such as positioning, or stylistic changes that aren't related to the overall design of the site.

- The `style` property isn't what you think it is. Let me explain. The `style` property itself is a `CSSStyleDeclaration` object representing all the various CSS properties for the given element. What you may not realize, however, is that the `style` property only provides access to CSS properties declared inline in the element's `style` attribute. That's it. Nothing else. It *doesn't* contain any CSS properties from the cascades of additional style sheets or properties inherited from parent elements. This means you can't use it to retrieve the overall computed style of the element. For that, you'll have to use a few different objects that we'll discuss later in this chapter.

If you thought that the `style` *property contained the computed style of the element, you're not alone. This is a common assumption that most people get wrong. Some people assume that the style sheet hasn't loaded yet or that some other browser-loading quirk is interfering with the* `style` *property, which simply isn't true. By design, the* `style` *property will never give you that information. It's a good example of why it's smart to spend a little time reading up on specifications so that you can avoid the frustration of spending hours fighting a "bug" that's really a feature.*

Keeping these two things in mind, let's look at the `style` property and some of its other quirks.

As you've already seen, you can set simple properties, such as the foreground color of the element, using the CSS `color` property:

```
element.style.color = 'red';
```

And you can set the element's `background-color` using

```
element.style.backgroundColor = 'red';
```

All the CSS properties follow this same pattern, but you'll notice that the `background-color` example uses a camel case syntax rather than the more familiar hyphenated CSS syntax. Converting `background-color` to the camel case `backgroundColor` syntax—by removing the hyphen and upper-casing the first letter of the following word—is necessary, because you're actually using a shortcut. To properly access style properties, the DOM2 Style `CSSStyleDeclaration` object has methods such as `setProperty()`, which use the proper, hyphenated CSS property name and value:

```
element.style.setProperty('background-color','red');
```

Again, the problem is that nonstandard browsers such as Microsoft Internet Explorer don't support the `setProperty()` method, so you'll have to rely on the shortcut method for cross-browser compatibility.

To keep things agnostic, add the following `setStyleById()` method to your ADS library along with the `setStylesByClassName()` and `setStylesByTagName()` methods. I've set up these methods to use a JavaScript object in the second parameter, so you can use JavaScript object notation to define multiple styles with proper CSS property names, though it will also work with the camel cased version as well:

```
ADS.setStyleById('example'{
    'background-color': 'red',
    'border': '1px solid black',
    'padding': '1em',
    'margin':'1em'
});
```

Within these setStyle methods, the hyphenated version of the CSS properties will be converted to camel case as necessary using the ADS.camelize() method you added when writing the HTML to DOM conversion tool in Chapter 3:

```
(function(){

window['ADS'] = {};

... above here is your existing library ...

/* changes the style of a single element by id */
function setStyleById(element, styles) {
    // Retrieve an object reference
    if(!(element = $(element))) return false;
    // Loop through  the styles object and apply each property
    for (property in styles) {
        if(!styles.hasOwnProperty(property)) continue;

        if(element.style.setProperty) {
            // DOM2 Style method
            element.style.setProperty(
            uncamelize(property,'-'),styles[property],null);
        } else {
            // Alternative method
            element.style[camelize(property)] = styles[property];
        }
    }
    return true;
}
window['ADS']['setStyle'] = setStyleById;
window['ADS']['setStyleById'] = setStyleById;

/* changes the style of multiple elements by class name */
function setStylesByClassName(parent, tag, className, styles) {
    if(!(parent = $(parent))) return false;
    var elements = getElementsByClassName(className, tag, parent);
    for (var e = 0 ; e < elements.length ; e++) {
        setStyleById(elements[e], styles);
    }
    return true;
}
window['ADS']['setStylesByClassName'] = setStylesByClassName;
/* changes the style of multiple elements by tag name */
function setStylesByTagName(tagname, styles, parent) {
    parent = $(parent) || document;
    var elements = parent.getElementsByTagName(tagname);
    for (var e = 0 ; e < elements.length ; e++) {
        setStyleById(elements[e], styles);
    }
}
window['ADS']['setStylesByTagName'] = setStylesByTagName;

... below here is your existing library ...

})();
```

These methods also provide a nice browser-agnostic way of modifying the style properties of related elements by class name:

```
ADS.setStylesByClassName(
    findClass',
    '*',
    document,
    {'background-color':'red'}
);
```

or by tag name

```
ADS.setStylesByTagName('a',{'text-decoration':'underline'});
```

But again, changing things like the `background-color` in your script embeds the style in your code. Even if you have no intention of sharing and your code is for you and you alone, it's still a good idea to refrain from mixing CSS style with your DOM scripts. When you're ready to move on from your current design and the time comes to edit your CSS, you'll be much happier messing with a few CSS files rather than sifting through all your scripts line by line, looking for every instance of `background-color=red`.

The only time mixing style and scripting is really acceptable is in the case of positioning, where the point of things like drag-and-drop interaction is to move the elements around the page. Setting `absolute` positioning along with the position, as follows, makes sense in the script, because the positioning and location are not part of the visual design, they're directly based on a reaction to the mouse movement on the page:

```
ADS.setStyleById('example',{
    'position':'absolute',
    'top':'10px',
    'left':'20px'
});
```

Font manipulation and color changes, on the other hand, don't belong in your scripts—at least not directly.

Switching styles based on a className

For small- to medium-scale presentation changes, such as colors, borders, backgrounds, and font styles that affect a few elements, you can avoid trudging through the code to look for style properties by implementing `className` switching. Simply use your DOM script to modify the `className` property of the desired elements, thus allowing you to alter the appearance based on predefined rules in your style sheets.

From an implementation point of view, this will require the manual or dynamic addition of a style sheet along with your script, increasing the usability and friendliness for designers. In addition, it also allows for reuse of the same DOM script across multiple designs, because your site's design isn't part of the DOM script. For example, instead of modifying the element's `background-color` directly through the `style` property, you could define two classes in your CSS file:

```
.normal {
     background-color: black;
}
.normal.modified {
     background-color: red;
}
```

And modify the `className` property of the element to apply the modified style:

```
var element = ADS.$('example');
element.className += ' modified';
```

Using common classes with className switching

When implementing `className` switching, it's a good idea to come up with a set of common classes that are only used to indicate changes in the document. These classes can act as pseudo-selectors and should only be used in combination with other selectors to trigger changes based on DOM scripts. For example, you may decide that `hover` is a reserved word. If a `hover` class was declared directly, as follows

```
.hover {
    position:absolute:
    top:100px;
    left:200px;
}
```

it would destroy the effectiveness of the DOM script. But if used only in combination with other selectors, like this

```
#cart.hover {
    background-color:yellow;
}
#sidebar.hover a {
    text-decoration:underline;
}
```

the hover class will act as a pseudo-class, in the same way as a CSS `a:hover` pseudo-class.

Using the className switching approach now maintains the proper separation and opens the presentation to CSS designers using the site's style sheets.

To make this a little easier, add these className manipulation methods to your ADS library:

```
(function(){

window['ADS'] = {};

... above here is your existing library ...

/* retrieve the classes as an array */
function getClassNames(element) {
    if(!(element = $(element))) return false;
    // Replace multiple spaces with one space and then
    // split the classname on spaces
    return element.className.replace(/\s+/,' ').split(' ');
};
window['ADS']['getClassNames'] = getClassNames;

/* check if a class exists on an element */
function hasClassName(element, className) {
    if(!(element = $(element))) return false;
    var classes = getClassNames(element);
    for (var i = 0; i < classes.length; i++) {
        // Check if the className matches and return true if it does
        if (classes[i] === className) { return true; }
    }
    return false;
};
window['ADS']['hasClassName'] = hasClassName;

/* Add a class to an element */
function addClassName(element, className) {
    if(!(element = $(element))) return false;
    // Append the classname to the end of the current className
    // If there is no className, don't include the space
    element.className += (element.className ? ' ' : '') + className;
    return true;
};
window['ADS']['addClassName'] = addClassName;

/* Remove a class from an element */
function removeClassName(element, className) {
    if(!(element = $(element))) return false;
    var classes = getClassNames(element);
    var length = classes.length
```

```
        // Loop through the array deleting matching items
        // You loop in reverse as you're deleting items from
        // the array which will shorten it.
        for (var i = length-1; i >= 0; i--) {
            if (classes[i] === className) { delete(classes[i]); }
        }
        element.className = classes.join(' ');
        return (length == classes.length ? false : true);
    };
    window['ADS']['removeClassName'] = removeClassName;

    ... below here is your existing library ...

    })();
```

Looking at the methods, they simply manipulate the className as appropriate:

- ADS.getClassNames(element) returns an array of the class names associated with the element.
- ADS.hasClassName(element, class) checks to see if the class is defined in the elements class name.
- ADS.addClassName(element, class) appends a class to className.
- ADS.removeClassName(element, class) removes the given class from the elements class name.

You can use these methods to easily retrieve all the classes associated with an element:

```
var element = document.getElementById('example');
var classes = ADS.getClassNames(element);
var class;
for (var i=0; class=classes[i]; i++) {
     //do something with the class
}
```

or to check if a class is assigned to the element before adding it, to produce a toggle effect:

```
function toggleClassName(element,className) {
    if(!ADS.hasClassName(element,className)) {
        ADS.addClassName(element,className);
    } else {
        ADS.removeClassName(element,className);
    }
}
ADS.addEvent('toggleButton','click',function() {
    toggleClassName(this,'active');
});
```

You can even reproduce a rollover effect with the appropriate event listeners and a hover class:

```
var element = document.getElementById('example');
ADS.addEvent(element,'mouseover',function() {
    ADS.addClassName(this,'hover');
});
ADS.addEvent(element,'mouseout',function() {
    ADS.removeClassName(this,'hover');
});
```

Drawbacks of using className switching

Using dynamic class names to indicate style changes is great, but it does have some drawbacks.

The first drawback is that the style of an element can influence the interaction in your scripts. Styles declared with different padding and margins can change the size and position of the elements, so your fancy draggable objects may no longer line up properly. Creating a common markup scheme for interactive elements can solve this problem. For example, you might create a policy where all draggable elements have a secondary inner container with a common class that CSS designers are free to manipulate, leaving the outer container to be positioned as expected.

The second drawback is shown in the earlier example when you appended `modified` to the `className` without checking to see what was already there. In most cases, you'll be using a more meaningful class name based on the desired action, but you'll also be adding and removing multiple class names for different actions. This will involve some checking to first determine what's already in the `className` so that you don't clutter it unnecessarily. Likewise, when you want to remove the `modified` class name, some string manipulation will be required to make sure you're removing the correct portion of the `className` string. Using the class name methods you added to your ADS library will help solve that problem.

Why not use setAttribute for class names?

At this point, you may be wondering why the examples have all been modifying the `className` property directly rather than using the `setAttribute()` method. The `className` refers to the class attribute of the `HTMLElement`, so there are actually three different ways you could possibly define the class, but they each work differently depending on the browser:

- `element.setAttribute('class','newClassName')` works in all W3C-compliant browsers.
- `element.setAttribute('className','newClassName')` works in Internet Explorer but isn't a W3C-compliant way of setting the value.
- `element.className = 'newClassName'` works in all browsers.

The third method works in all browsers and is a completely valid way of altering an `HTMLElements` `className` read/write property, so we'll stick to it for ease of use.

Switching the style sheet

For changes on a much grander scale, such as changing the layout of the entire page, it doesn't make sense to iterate over the DOM tree and manually modify the style of each individual element. The easiest thing to do would be to simply switch the active style sheet for a completely different one. This can be done several ways depending on how you've set up your CSS style sheets and rules:

- You can use the `rel` property of the `<link>` element to define alternative style sheets and switch among them accordingly.
- You can apply a class to the body tag and modify your CSS selectors depending on the body class—effectively `className` switching using the body tag as the root.
- You can dynamically add and remove style sheets.

Regardless of which method you choose, the desired effect is to drastically change the layout and presentation of most, if not all, of the elements in the document.

Using alternative style sheets

Switching among alternative style sheets was originally proposed by Paul Sowden in his article "Alternative Style: Working with Alternate Style Sheets" (`http://alistapart.com/stories/alternate/`). This method takes advantage of some of the features of the DOM2 HTML `<link>` element.

If you've been following the separation of presentation and markup methodologies, you'll already be familiar with the `<link>` element. It's used in the head of your document to include your style sheet:

```
<link type="text/css" href="/path/to/style.css" media="screen" />
```

In a `<link>` element

- `type` is used to indicate the MIME type of the style file, in this case `text/css`.
- `href` is used to specify the location of the style sheet.
- `media` limits the types of devices that should implement the style sheet. However, it's up to the device and software to check the media types. Some devices will ignore the media or use the incorrect one.

These attributes are the most common ones you've dealt with, but there are others. Link elements also include the following attributes (among others):

- `rel` indicates the relationship between the style sheet and the document.
- `disabled` indicates if the style sheet is active.
- `title` is a title associated with the style sheet.

You'll notice that these attributes are similar to the `CSSStyleSheet` *properties form the beginning of this chapter.*

You can use the `rel="stylesheet"` attribute to specify if a style sheet should be applied to a document immediately:

```
<link rel="stylesheet" type="text/css" ➥
href="/path/to/style.css" media="screen">
```

or if it should be considered an alternate style sheet and disabled by default using `rel="alternate stylesheet"`:

```
<link rel="alternate stylesheet" type="text/css" ➥
href="/path/to/style.css" media="screen">
```

When the relationship is `alternate stylesheet` the browser will load the style sheet, but it will also set the `disabled` flag to `true`, so it won't be applied to the document immediately.

The `title` attribute has no effect on the style sheet itself but you can take advantage of it in your script. You can use it as an identifier to dynamically create the list of available styles, allowing the user to switch among them at will. If you take a look at the example in the `chapter5/switcher/example.html` source, I've included a few example CSS layouts, so you can see the alternate style sheets in action:

```
<!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>Style Switcher</title>
    <!--
        The common styles that won't change.
        Leave the title off so they won't be included in the list
    -->
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="../chapter.css" />
    <link rel="stylesheet" type="text/css"
        href="common.css" media="screen">

    <!-- AdvancED DOM Scripting Simple Style (the default) -->
    <link rel="stylesheet" title="AdvancED DOM Scripting"
        type="text/css" href="ads.css" media="screen">
    <!--[if true]>
    <link rel="alternate stylesheet" title="AdvancED DOM Scripting"
        type="text/css" href="adsIE.css" media="screen">
    <![endif]-->

    <!-- A friends of ED style -->
    <link rel="alternate stylesheet" title="friends of ED"
        type="text/css" href="foed.css" media="screen">

    <!-- An Apress style -->
    <link rel="alternate stylesheet" title="Apress"
        type="text/css" href="apress.css" media="screen">


    <!-- ADS Library (full version from source linked here) -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>
    <!-- The load script -->
    <script type="text/javascript" src="styleSwitcher.js"></script>


</head>
<body>
<h1>Style Switcher</h1>
    <div id="content">
        <h2>It's Easy!</h2>
        <ul id="styleSwitcher"></ul>
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
        Quisque iaculis elit in mauris. Mauris euismod tempor tortor.
        Integer fringilla, orci at venenatis consequat, lorem ipsum.
        Morbi ornare sollicitudin justo. Nulla est. Cras lorem.</p>
    </div>
</body>
</html>
```

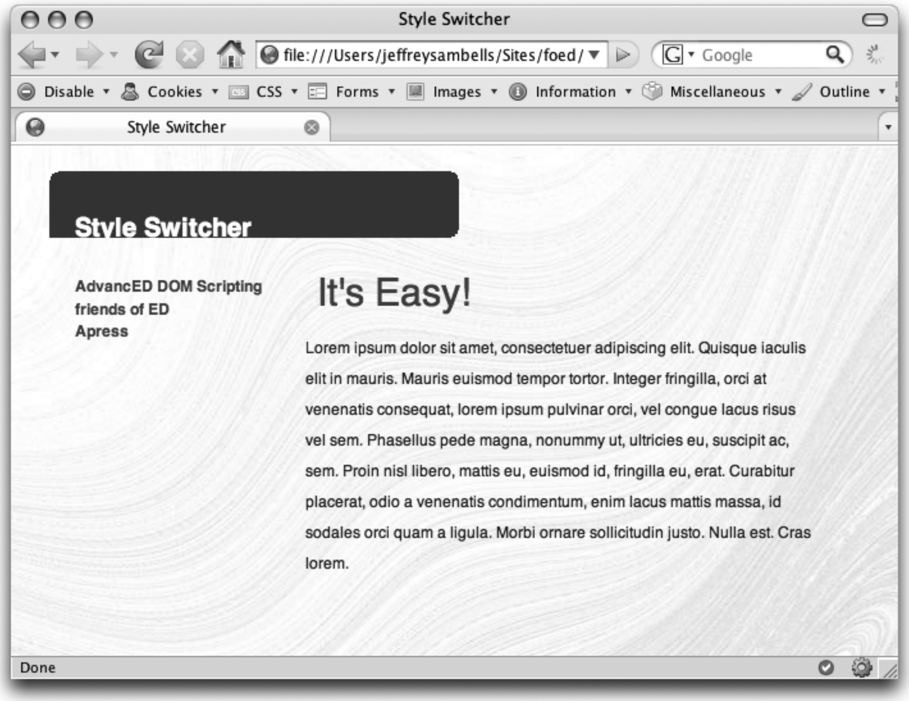
In this case, the common CSS files have no title, while the `friends of ED` and `Apress` styles each have one style sheet, and the `AdvancED DOM Scripting` style contains two (one just for Internet Explorer). The following empty list in the markup

```
<ul id="styleSwitcher"></ul>
```

is populated by the load event in `chapter5/switch/styleSwitcher.js` based on the available style sheets and their titles:

```
ADS.addEvent(window,'load',function() {
    // Retrieve all the link elements
    var list = ADS.$('styleSwitcher');
    var links = document.getElementsByTagName('link');
    var titles = [];

    for (var i=0 ; i<links.length ; i++) {

        // Skip <link> element that aren't styles with titles.
        if(links[i].getAttribute("rel").indexOf("style") != -1
            && links[i].getAttribute("title")) {

            // Append a new item to the list if the title hasn't
            // already been added
            var title = links[i].getAttribute("title");
            if(!titles[title]) {

                var a = document.createElement('A');
                a.appendChild(document.createTextNode(title));
                a.setAttribute('href','#');
                a.setAttribute('title','Activate ' + title);
                a.setAttribute('rel',title);
                ADS.addEvent(a,'click',function(W3CEvent) {
                    // When clicked activate the style sheet indicated
                    // by the title in the anchor's rel property
                    setActiveStyleSheet(this.getAttribute('rel'));
                    ADS.preventDefault(W3CEvent);
                });

                var li = document.createElement('LI');
                li.appendChild(a);

                list.appendChild(li);

                // Set the titles array to true for this title
                // so that it will be skipped if multiple sheets use
                // the same title
                titles[title] = true;
            }
        }
    }
});
```

styleSwitcher.js creates a regular list that you can style as needed, as shown in Figure 5-4.



Style Switcher

AdvancED DOM Scripting
friends of ED
Apress

It's Easy!

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Quisque iaculis elit in mauris. Mauris euismod tempor tortor. Integer fringilla, orci at venenatis consequat, lorem ipsum pulvinar orci, vel congue lacus risus vel sem. Phasellus pede magna, nonummy ut, ultricies eu, suscipit ac, sem. Proin nisl libero, mattis eu, euismod id, fringilla eu, erat. Curabitur placerat, odio a venenatis condimentum, enim lacus mattis massa, id sodales orci quam a ligula. Morbi ornare sollicitudin justo. Nulla est. Cras lorem.

Figure 5-4. The style switcher list populated from the link title attributes

The resulting list allows you to select among the alternative style sheet sets using the anchor's click event listener and the `setActiveStyleSheet()` function from Paul Sowden's original article, which follows:

```
function setActiveStyleSheet(title) {
   var i, a, main;
   for(i=0; (a = document.getElementsByTagName("link")[i]); i++) {
     if(a.getAttribute("rel").indexOf("style") != -1
        && a.getAttribute("title")) {
       a.disabled = true;
       if(a.getAttribute("title") == title) a.disabled = false;
     }
   }
}
```

The result is illustrated in Figure 5-5.

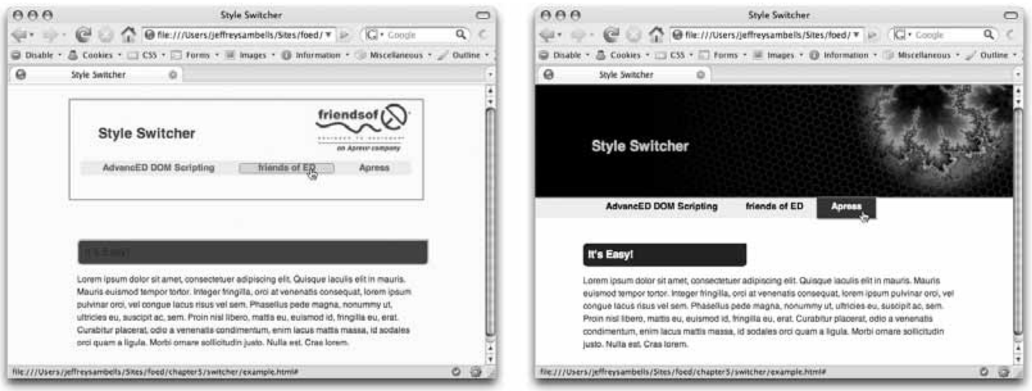


Figure 5-5. The styles as shown when clicking each link

This example is incomplete, as it doesn't maintain the style across multiple pages in your site, but it does illustrate the ability to enable and disable multiple style sheets.

Switching the body className

This idea follows the same principles of `className` switching that you saw earlier, but in this case, you're switching the `className` on the body tag. Andy Clarke and James Edwards describe the full body class switching technique in more detail in their article "Invasion of the Body Switchers" (`http://alistapart.com/articles/bodyswitchers`), but the definition of the CSS rules is the only real difference from the earlier `className` switching technique applied to elements. To apply styles based on the body tag, you simply include one style sheet that has the body's class selector in all the declarations:

```
/* common Style*/
body {
     font: 62.5%/1.2em sans-serif;
     color: #1a3800;
     text-align:center;
}

#container {
    text-align:left;
}

/* AdvancED DOM Scripting style */

body.ads {
    background-image: url(images/ads-bg.jpg);
}

body.ads h1 {
    height: 12px;
    margin: 1em 3em 0 30px;
    padding: 2em 0 0 20px;
    padding-bottom: 0em ;
    background: #f06;
    width: 300px;
    overflow: hidden;
    white-space: nowrap;
    -moz-border-radius: 0.5em;
    -moz-border-radius-bottomleft: 0em
}
body.ads h2 {
    border: 0;
    font-size: 3em;
    color: #1A5B9D;
    font-weight: normal;
}

/*  etc */

/* friends of ED style */

body.foed {
    font-family: sans-serif;
    background-color: #fffbf2;
    line-height: 1.8em;
}

body.foed h1 {
    height: 100px;
    color: #f06;
    padding: 40px 0 0 40px;
    background: transparent url(images/foed.png) no-repeat top right;
    border: 2px solid #999;
    width: 460px;
    margin: 1em auto 1em auto;
}

body.foed h2 {
    border-width: 2px;
    background: #f06;
    padding: 0.5em;
    width: 100%;
}

/*  etc */

/* Apress style */
body.apress {
    font-family: sans-serif;
    background-color: white;
    line-height: 1.8em;
}

body.apress h1 {
    height: 80px;
    background: black url(images/fractal.jpg) no-repeat top right;
    color: #ffCC00;
    padding: 80px 0 0 80px;
}

body.apress h2 {
    border: 0;
    background: #900;
    padding: 0.5em;
    color: white;
    width: 40%;
}
/*  etc */
```

Next, using the class name methods in your ADS library from earlier in this chapter, or some other function you create yourself, you can switch the body's class dynamically, and the CSS rules will alter the page accordingly:

```
ADS.addEvent('ads-anchor','click',function() {
    ADS.addClassName(document.body,'asd');
});
ADS.addEvent('foed-anchor','click',function() {
    ADS.addClassName(document.body,'foed');
});
ADS.addEvent('apress-anchor','click',function() {
    ADS.addClassName(document.body,'apress');
});
```

Automatically populating a list of styles using this method is a little tricky, as there's no easy way to associate a title with the different sets.

This class name and CSS cascade technique can be applied to any level of the document, not just individual elements and the body.

Dynamically loading and removing style sheets

The third technique involves dynamically loading and unloading the style sheets in your document and is pretty straightforward. All you have to do is create a new `<link>` through the `document.createElement()` method with the appropriate properties. Add these two methods to your ADS library, and they will do just that:

```
(function(){

window['ADS'] = {};

... above here is your existing library ...

/* Add a new stylesheet */
function addStyleSheet(url,media) {
    media = media || 'screen';
    var link = document.createElement('LINK');
    link.setAttribute('rel','stylesheet');
    link.setAttribute('type','text/css');
    link.setAttribute('href',url);
    link.setAttribute('media',media);
    document.getElementsByTagName('head')[0].appendChild(link);
}
window['ADS']['addStyleSheet'] = addStyleSheet;

/* Remove a stylesheet */
function removeStyleSheet(url,media) {
    var styles = getStyleSheets(url,media);
    for(var i = 0 ; i < styles.length ; i++) {
        var node = styles[i].ownerNode || styles[i].owningElement;
        // Disable the stylesheet
        styles[i].disabled = true;
        // Remove the node
        node.parentNode.removeChild(node);
    }
}
window['ADS']['removeStyleSheet'] = removeStyleSheet;

... below here is your existing library ...

})();
```

The `ADS.addStyleSheet()` method will add the style sheet at the given URL and media:

```
ADS.addStyleSheet('/path/to/style.css','screen');
```

Likewise, `ADS.removeStyleSheet()` will remove the style sheet for the given URL and media:

```
ADS.removeStyleSheet('/path/to/style.css','screen');
```

The `ADS.removeStyleSheet()` *method relies on the* `ADS.getStyleSheets()` *method that you'll be adding next.*

By loading and unloading various style sheets, you can switch among as many style sheets as you like.

Modifying CSS rules

When your scripts need to modify the appearance of several related elements on a page, it may be easier to modify the actual CSS rule in the style sheet itself rather than locating all the elements and modifying their individual style properties. This is exceptionally useful if you want a few specific properties to change but, at the same time, you still want the cascade of declarations to apply for things such as an anchor's `:hover` pseudo-class.

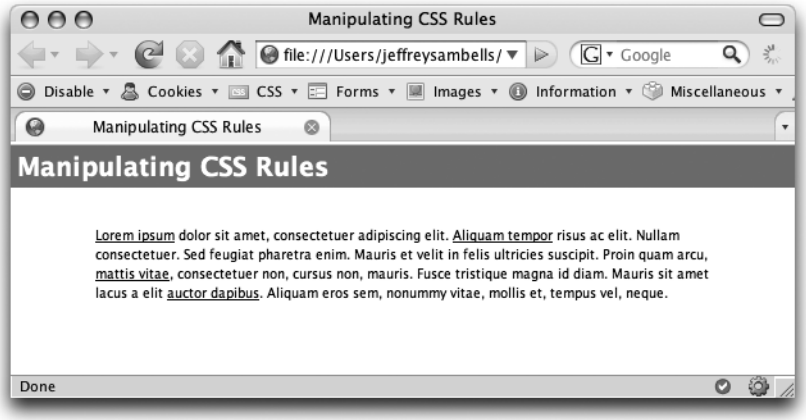
Take these simple anchor CSS declarations:

```
/* anchor styles */
a {
    font-weight:normal;
}
a:link {
    text-decoration:none;
    color: black;
}
a:visited {
    text-decoration:none;
}
a:hover {
    color: #248030;
    text-decoration: none;
}
```

If you have a few CSS rules applied to a simple paragraph of text with a few links, such as this

```
<p><a href="http://lipsum.com" title="Go to lipsum.com">Lorem ➥
ipsum</a> dolor sit amet, consectetuer adipiscing elit. <a ➥
href="http://lipsum.com" title="Go to lipsum.com">Aliquam ➥
tempor</a> risus ac elit. Nullam consectetuer. Sed feugiat ➥
pharetra enim. Mauris et velit in felis ultricies suscipit. ➥
Proin quam arcu, <a href="http://lipsum.com" title="Go to ➥
lipsum.com">mattis vitae</a>, consectetuer non, cursus non, ➥
mauris. Fusce tristique magna id diam. Mauris sit amet lacus a ➥
elit <a href="http://lipsum.com" title="Go to lipsum.com">auctor ➥
dapibus</a>. Aliquam eros sem, nonummy vitae, mollis et, tempus ➥
vel, neque.</p>
```

you would get something similar to the page in Figure 5-6.



Manipulating CSS Rules

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam tempor risus ac elit. Nullam consectetuer. Sed feugiat pharetra enim. Mauris et velit in felis ultricies suscipit. Proin quam arcu, mattis vitae, consectetuer non, cursus non, mauris. Fusce tristique magna id diam. Mauris sit amet lacus a elit auctor dapibus. Aliquam eros sem, nonummy vitae, mollis et, tempus vel, neque.

Figure 5-6. A page of text with various links

If you then want your DOM script to reveal the URL behind every anchor on the page, you could take several approaches.

First, you could loop through all the links on the page and modify each element as appropriate using DOM methods to read the `href` property and append the appropriate content into the anchor. That would work, but it will require the script to search the DOM tree for all the appropriate anchor tags and modify the markup directly. If the browser supports a few CSS2 properties and selectors, such as `content` and the `:after` pseudo-class, it can be easily done with a simple CSS rule:

```
a[href]:after {
    content: " (" attr(href) ") ";
    font-size: 40%;
    color: #16009b;
}
```

As an alternative to altering the markup of your DOM document, you can use your DOM script to add or edit the CSS rules in your style sheet—but it gets a little tricky. Your document can contain several style sheets in the `document.styleSheets` property. Unless you know which style sheet you want to manipulate, you'll have to loop through them all looking for the appropriate selectors. Even if you do know the URL of the style sheet you want, you'll still have to loop through them all to figure out which one it is, as the `document.styleSheets` list is indexed numerically. Here's a method you can add to your ADS library to help sort out the `document.styleSheets` list by looking for the appropriate `href` and `media` properties:

```
(function(){

window['ADS'] = {};

... above here is your existing library ...
/* Retrieve an array of all the stylesheets by URL */
function getStyleSheets(url,media) {
    var sheets = [];
    for(var i = 0 ; i < document.styleSheets.length ; i++) {
        if (url && document.styleSheets[i].href.indexOf(url) == -1) {
            continue;
        }
        if(media) {
            // Normalize the media strings
            media = media.replace(/,\s*/,',');
            var sheetMedia;

            if(document.styleSheets[i].media.mediaText) {
                // DOM mehtod
                sheetMedia = document.styleSheets[i].media.➥
mediaText.replace(/,\s*/,',');
                // Safari adds an extra comma and space
                sheetMedia = sheetMedia.replace(/,\s*$/,'');
            } else {
                // MSIE
                sheetMedia = document.styleSheets[i].media.➥
replace(/,\s*/,',');
            }
            // Skip it if the media don't match
            if (media != sheetMedia) { continue; }
        }
        sheets.push(document.styleSheets[i]);
    }
    return sheets;
}
window['ADS']['getStyleSheets'] = getStyleSheets;

... below here is your existing library ...

})();
```

This getStyleSheets() method returns an array of style sheets that match the given href and optional media properties of the CSSStyleSheet objects. In the case of an inline <style> block in the head or body of your document—which you shouldn't be doing—the href will be that of the current page.

Now you can find the style sheets you need and modify them by adding these ADS.editCSSRule() and ADS.addCSSRule() methods to your ADS library:

```
(function(){

window['ADS'] = {};

... above here is your existing library ...
```

```
/* Edit a CSS rule */
function editCSSRule(selector,styles,url,media) {
    var styleSheets = (typeof url == 'array' ? url : ➥
getStyleSheets(url,media));

    for ( i = 0; i < styleSheets.length; i++ ) {

        // Retrieve the list of rules
        // The DOM2 Style method is styleSheets[i].cssRules
        // The MSIE method is styleSheets[i].rules
        var rules = styleSheets[i].cssRules || styleSheets[i].rules;
        if (!rules) { continue; }

        // Convert to uppercase as MSIE defaults to UPPERCASE tags.
        // this could cause conflicts if you're using case
        // sensitive ids
        selector = selector.toUpperCase();

        for(var j = 0; j < rules.length; j++) {
            // Check if it matches
            if(rules[j].selectorText.toUpperCase() == selector) {
                for (property in styles) {
                    if(!styles.hasOwnProperty(property)) { continue; }
                    // Set the new style property
                    rules[j].style[camelize(property)] = ➥
styles[property];
                }
            }
        }
    }
}
window['ADS']['editCSSRule'] = editCSSRule;

/* Add a CSS rule */
function addCSSRule(selector, styles, index, url, media) {
    var declaration = '';

    // Build the declaration string from the style object
    for (property in styles) {
        if(!styles.hasOwnProperty(property)) { continue; }
        declaration += property + ':' + styles[property] + '; ';
    }

    var styleSheets = (typeof url == 'array' ? url :➥
getStyleSheets(url,media));
    var newIndex;
    for(var i = 0 ; i < styleSheets.length ; i++) {
        // Add the rule
        if(styleSheets[i].insertRule) {
            // The DOM2 Style method
            // index = length is the end of the list
            newIndex = (index >= 0 ? index :➥
styleSheets[i].cssRules.length);
            styleSheets[i].insertRule(
                selector + ' { ' + declaration + ' } ',
                newIndex
            );
        } else if(styleSheets[i].addRule) {
            // The Microsoft method
            // index = -1 is the end of the list
            newIndex = (index >= 0 ? index : -1);
            styleSheets[i].addRule(selector, declaration, newIndex);
        }
    }
}
window['ADS']['addCSSRule'] = addCSSRule;

... below here is your existing library ...

})();
```

Remember, when editing rules you can only edit a rule that has been explicitly declared in the style sheet. Also, the `ADS.addCSSRule()` *method doesn't work properly in Safari, but it does work in the latest WebKit release from Webkit.org, so you can expect it to work in a future version of Safari.*

Applying these methods to the previous page of text and links, you can make simple modifications, such as highlighting all the anchors with a background color:

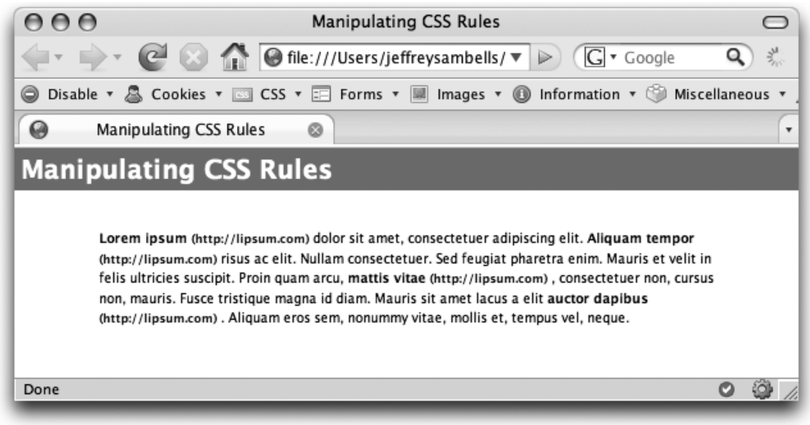
```
ADS.editCSSRule('a',{'background-color':'yellow'});
```

When editing the anchor background color, all the anchors will receive this new color unless another style in the cascade overrides the color. If, for example, `a:hover` *was declared with a different background color, the hover state would still work as expected. Modifying the style property of the element itself can't implement changes such as this.*

You can also add a new selector to the style sheet to reveal the URL, as shown in Figure 5-7, but this will only work in CSS 2.1–compatible browsers such as Firefox, Safari, and Opera:

```
ADS.addCSSRule('a[href]:after',{
    'content':'" (" attr(href) ") "',
    'font-size': '40%',
    'color': '#16009b'
});
```

In the case of the ADS.addCSSRule() method, the third index parameter specifies where in the CSS file the rule should be added, with null representing the end of the file.



Manipulating CSS Rules

Lorem ipsum (http://lipsum.com) dolor sit amet, consectetuer adipiscing elit. **Aliquam tempor** (http://lipsum.com) risus ac elit. Nullam consectetuer. Sed feugiat pharetra enim. Mauris et velit in felis ultricies suscipit. Proin quam arcu, **mattis vitae** (http://lipsum.com) , consectetuer non, cursus non, mauris. Fusce tristique magna id diam. Mauris sit amet lacus a elit **auctor dapibus** (http://lipsum.com) . Aliquam eros sem, nonummy vitae, mollis et, tempus vel, neque.

Figure 5-7. Anchors altered to show the href attribute

The page of example text only contains a single style sheet, so the optional url and media parameters for the edit and add methods aren't necessary. If you have multiple style sheets, you can retrieve just those matching url by including an additional parameter:

```
ADS.editCSSRule(
    'a:hover',
    {'text-decoration':'underline'},
    '/path/to/style.css'
);
ADS.addCSSRule(
    'a',
    {'font-weight':'bold'},
    null,
    '/path/to/style.css'
);
```

Modifying each individual style property, switching the body class, and editing the CSS rules each have their advantages, so you'll have to decide which solution is best for your application.

AdvancED image replacement revisited

Earlier in this chapter you created an image replacement script in chapter5/image-replacement/advancED.html example that required you to hard-code the URL of the image into the load event:

```
ADS.addEvent(window, 'load', function() {

    ... cut ...

    // Load the image
    // This hardcoded path isn't ideal.
    image.src = 'http://advanceddomscripting.com/images/➥
advancED-replace.png';
});
```

Rather than hard-code the URL into the DOM script, you can specify a style sheet for the image replacement, for example advancED.css, and then search its rules for the required background URL.

To do so, you create a CSS file that follows the same logic of the earlier CSS file, as shown in the chapter5/image-replacement/advancED.css source:

```
/* The styling for the text if images are disabled */
#advancedHeader {
    color: #1A5B9D;
}

/* Sizing the header for the image */
#advancedHeader.advancED {
    height:60px;
    position:relative;
    overflow:hidden;
}
/* Hide the text with the opaque image*/
#advancedHeader.advancED span {
    background: white url(http://advanceddomscripting.com/➥
images/advancED-replace.png) no-repeat;
    display:block;
    width:100%;
    height:100%;
    position:absolute;
}
```

Next, to retrieve the URL of the image, you'll need to examine the selectorText attribute of each rule in the style sheet by locating the ID number and advancED class combination along with the <span> tag:

```
#advancedHeader.advancED span
```

Again, it's a little tricky. This rule could also be defined as

```
.advancED#advancedHeader span
```

or with the a tag, such as a <h2>

```
h2.advancED#advancedHeader span
```

Also, in Internet Explorer, the rule's selectorText property will be converted to .class#id and the tags converted to uppercase, regardless of how it's written in the CSS file:

```
.advancED#advancedHeader SPAN
```

You can repurpose the earlier image-replacement load event into an auxiliary method and simply pass in the ID of the element, as shown in the chapter5/image-replacement-revisited/advancED.js source:

```
function replaceImage(element) {
    // Retrieve the element
    var element = ADS.$(element);
    // Create an image element
    var image = document.createElement('IMG');

    // Only add the span and class if the image loads
    ADS.addEvent(image, 'load', function() {

        var s = document.createElement('SPAN');
        // Prepend the span to the element's children
        ADS.prependChild(element,s);

        // Create the title attribute as necessary
        if(!element.getAttribute('title')) {
            var i, child;
            var title = '';
            // Loop through the children and assemble the title
            for(i=0 ; child = element.childNodes[i] ; i++ ) {
                if(child.nodeValue) title += child.nodeValue;
            }
            element.setAttribute('title',title);
        }
        // Modify the class name to indicate the change
        // and apply the CSS
        ADS.addClassName(element,'advancED');
    });


    // Load the image
    var styleSheet = ADS.getStyleSheets('advancED.css')[0];
    if(!styleSheet) return;

    var list = styleSheet.cssRules || styleSheet.rules
    if(!list) return;

    var rule;
    for(var j = 0 ; rule = list[j] ; j++) {

        // Look for the rule:
        // either: #element-id.advancED span
        // or .advancED#element-id span
        // or as in MSIE: .advancED#element-id SPAN
        // where element-id is the one passed into this method
```

```
                if(
                    rule.selectorText.indexOf('#' + ➥
element.getAttribute('id')) !== -1
                    && rule.selectorText.indexOf('.advancED') !== -1
                    && rule.selectorText.toUpperCase().indexOf(' SPAN') !== -1
                ) {
                    // look for a url() in the css using
                    // the regex: /url\(([^\)]+)\)/
                    var matches = rule.style.cssText.match(/url\(([^\)]+)\)/);
                    // matches[1] will contain the value in the
                    // capturing parenthesis of the regex
                    if(matches[1]) {
                        image.src = matches[1];
                        break;
                    }
                }
            }
        }

ADS.addEvent(window, 'load', function() {
    replaceImage('advancedHeader');
});
```

With this load event and replaceImage() method, you achieve the same result by retrieving the advancED.css file and using a regular expression to extract the URL from the appropriate rule. The style of the object is now completely separated from the DOM script, and the CSS is free to change as necessary.

To take it one step further, you can make the DOM script even more unobtrusive by adding an additional class to images your markup:

```
<h2 id="advancedHeader" class="replaceMe">Advanced DOM Scripting</h2>
```

and use the load event to look for any element that match that class and replace them as required:

```
ADS.addEvent(window, 'load', function() {
    var replacements = ADS.getElementsByClassName('replaceMe');
    for(var i=0 ; i< replacements.length ; i++) {
        replaceImage(replacements[i]);
    }
});
```

This last addition makes the script maintenance free, as all you need to do is include the script and mark up your document and CSS accordingly. No additional DOM script editing is necessary.

Accessing the computed style

Before you modify the presentation of an element, you'll want to first determine its current style properties. As you saw, an element's style property only applies to styles defined inline, so you can't use it to retrieve the *computed* style. You could retrieve additional style information from the CSS rules themselves, but it would be a long and complicated process better left to the browser. If you want to access all the computed stylistic properties of an element, as determined by the cascade, you'll need to look at alternative properties.

DOM2 Style includes a method in the document.defaultView called getComputedStyle() for just this purpose. This method returns a read-only CSSStyleDeclaration object with all the computed style properties for the given element, not just those defined inline.

After retrieving the computed style for a given element, you can retrieve the style information the same way you did from the element's style property:

```
var element = ADS.$('example');
var styles = document.defaultView.getComputedStyle(element);
```

Retrieving the background-color is as simple as this:

```
var color = styles.getProperty('background-color');
```

The problem with this is, again, that Microsoft has its own version using the element's currentStyle property, so you'll have to access the proprietary methods as well using the following ADS.getStyle() method in your ADS library:

```
(function(){

window['ADS'] = {};

... above here is your existing library ...

/* retrieve the computed style of an element */
function getStyle(element,property) {
    if(!(element = $(element)) || !property) return false;
    // Check for the value in the element's style property
    var value = element.style[camelize(property)];
    if (!value) {
        // Retrieve the computed style value
        if (document.defaultView && document.defaultView.➥
getComputedStyle) {
            // The DOM method
            var css = document.defaultView.getComputedStyle(
                element, null
            );
            value = css ? css.getPropertyValue(property) : null;
        } else if (element.currentStyle) {
            // The MSIE method
            value = element.currentStyle[camelize(property)];
        }
    }
    // Return an empty string rather than auto so that you don't
    // have to check for auto values
    return value == 'auto' ? '' : value;
}
window['ADS']['getStyle'] = getStyle;
window['ADS']['getStyleById'] = getStyle;

... below here is your existing library ...

})();
```

The Microsoft filter property

I'm not one to promote proprietary features that aren't standard across browsers—except when you're using them to fix the inconsistencies in the browser they're in. In this case, I'm referring to the Internet Explorer `filter` property and the ability to "fix" transparent PNG files in Microsoft Internet Explorer version 6 or less.

The object I'm about to present doesn't follow any standards and is only expected to run in Internet Explorer version 6 or less, so the majority of the code is Microsoft-specific. If the code encounters anything greater than Internet Explorer 6 or another browser, it won't run, as it has no need to.

The problem in Internet Explorer 6 is that the translucent areas of a PNG file don't render properly; instead, they appear as a light blue box, as shown in Figure 5-8.

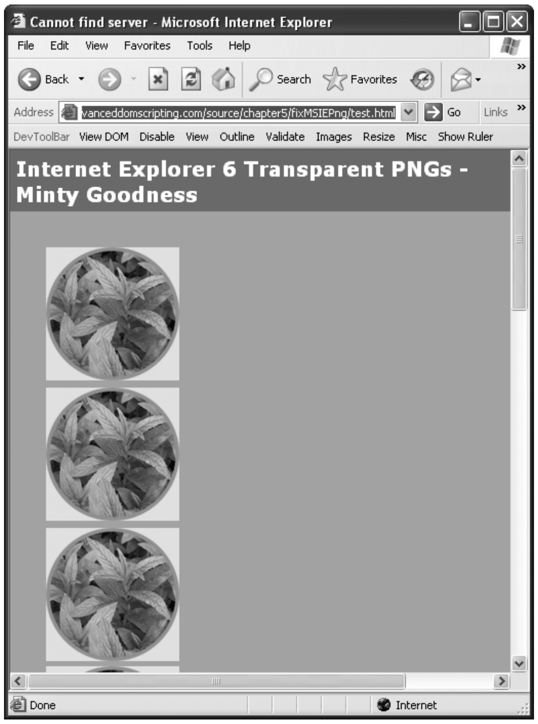

Cannot find server - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address vanceddomscripting.com/source/chapter5/fixMSIEPng/test.html Go Links
DevToolBar View DOM Disable View Outline Validate Images Resize Misc Show Ruler
Internet Explorer 6 Transparent PNGs - Minty Goodness
Done Internet

Figure 5-8. Translucent PNG files as they appear in Internet Explorer

This problem can crop up in places such as these:

- An `<img>` element with a PNG as its source
- Style sheets imported using the `<link>` element or `@import` rules that use translucent PNGs in the background of the elements
- Inline style attributes that define translucent PNGs as the background of the element

To work around this problem, you can use the `Microsoft.AlphaImageLoader` filter, which allows you to use a translucent PNG file as the background of your HTML elements. This can be applied directly in your CSS file by using conditional comments to include a Microsoft-only CSS file:

```
<!--[if lte IE 6]>
    <link rel="stylesheet" href="style-lte-IE-6.css" type="text/css"/>
<![endif]-->
```

and including the appropriate rules for the specific element:

```
#example {
    /* Internet Explorer translucent PNG hack */
    background-color: transparent;
    background-image: url(blank.gif);
    width: 100px;
    height: 100px;
    /* filter requires the full path */
    filter: progid:DXImageTransform.Microsoft.AlphaImageLoader(➥
src="translucent-image.png", sizingMethod="scale");
}
```

The example `chapter5/fixMSIEPng/test.html` source file includes this `fixMSIEPng()` method in `chapter5/fixMSIEPng/fixMSIEPng.js` that runs when the page is loaded and walks through the document and CSS files looking for PNG related files to fix:

```
function fixMSIEPng() {
    if(!document.body.filters) {
        // Not MSIE
        return;
    }
    if(7 <= parseFloat(navigator.appVersion.split("MSIE")[1])) {
        // 7+ supports PNG
        return;
    }
    // Fix the inline images
    if(document.images) {
        var images = document.images;
        var img = null;

        for(var i=images.length-1; img=images[i]; i--) {

            // Check if it's PNG image
            if(img.src
                && img.src.substring(
                    img.src.length-3,
                    img.src.length
                ).toLowerCase() !== 'png'
            ) {
                // Skip it
                continue;
            }

            // Build the style property for the outer element
            var inlineStyle = '';
            if (img.align == 'left' || img.align == 'right') {
                inlineStyle += 'float:' + img.align + ';';
            }

            if (img.parentElement.nodeName == 'A') {
                // This image is inside an anchor so show a hand
                inlineStyle += 'cursor:hand;';
            }

            // Make the display inline-block so that it can have a
            // width and height yet still be positioned properly
            inlineStyle += 'display:inline-block;';

            // Grab any other CSS style applied to the element
            if(img.style && img.style.cssText) {
                inlineStyle += img.style.cssText;
            }

            // Wrap a <span> around the image with the appropriate
            // style and information such as className and ID
            img.outerHTML = '<span '
            + (img.id ? ' id="' + img.id + '"' : '' )
            + (img.className ? 'class="' + img.className + '" ' : '')
            + ' style="width:' + img.width + 'px; height:'
            + img.height + 'px;'
            + inlineStyle
            + ';filter:progid:DXImageTransform.Microsoft'
            + '.AlphaImageLoader(src=\''
            + img.src
            + '\', sizingMethod=\'scale\');"></span>';

        }
    }

    // Create a private method to apply in the next set of loops
    // This sets the appropriate styles for the elements
    function addFilters(e) {
        // Check if the element has style, a background and verify
```

```
        // it doesn't already have a filter applied
        if(
            e.style
            && e.style.background
            && !e.style.filter
        ) {
            // Check if it's a PNG
            var src=null;
            if(src = e.style.backgroundImage.➥
match(/^url\((.*\.png)\)$/i)) {
                e.style.backgroundColor = 'transparent';
                e.style.backgroundImage = 'url()';
                e.style.filter = 'progid:DXImageTransform.Microsoft.'
                    + 'AlphaImageLoader(src=\''
                    + src[1]
                    + '\',sizingMethod=\''
                    + (( e.style.width && e.style.height ) ? ➥
'scale' : 'crop' )
                    + '\')';
            }
        }
    }

    // Create a private recursive processing method to apply the
    // addFilters() method to the style sheets
    function processRules(styleSheet) {
        for (var i in styleSheet.rules) {
            addFilters(styleSheet.rules[i]);
        }

        //recurse for @import stylesheets...
        if(styleSheet.imports) {
            for (var j in styleSheet.imports) {
                processRules(styleSheet.imports[j]);
            }
        }
    }

    // Process each style sheet
    var styleSheets = document.styleSheets;
    for(var i=0; i < styleSheets.length; i++) {
        processRules(styleSheets[i]);
    }

    // Fix the inline style properties
    if(document.all) {
        var all = document.all;
```

```
            for(var i=0; i < all.length; i++) {
                addFilters(all[i]);
            }
        }

    }
    if(window.attachEvent) window.attachEvent("onload", fixMSIEPng);
```

The function will only run in Internet Explorer, as it uses the Internet Explorer–only attachEvent method, and it will transform all the PNGs as shown in Figure 5-9.

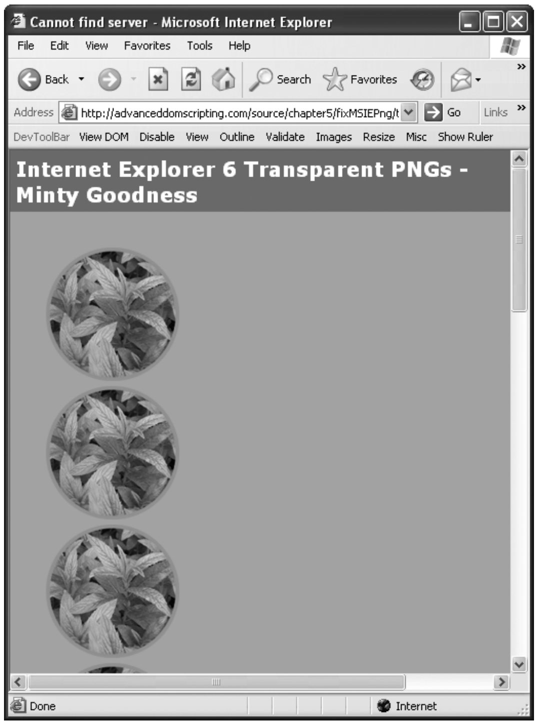

Cannot find server - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://advanceddomscripting.com/source/chapter5/fixMSIEPng/t Go Links
DevToolBar View DOM Disable View Outline Validate Images Resize Misc Show Ruler
Internet Explorer 6 Transparent PNGs - Minty Goodness
Done Internet

Figure 5-9. The PNGs with translucent backgrounds after the fix has run

Please note that the filter property introduces a bug: in some cases, anchors and other clickable elements become inoperative within an element with a filter applied. In the following example, the nested anchor may not work if #background has a filter style applied:

```
<div id="background">
    <div id="content">
        <p>The content with <a href="http://example.com">anchors</a></p>
    </div>
</div>
```

To avoid this, I suggest modifying the markup to something similar to the following, and using CSS to layer the #content above the translucent #background with the filter applied:

```
<div id="background"></div>
<div id="content">
  <p>The content with <a href="http://example.com">anchors</a></p>
</div>
```

Practical example: a simple transition effect

Before I go further, this next example comes with a caution: transition effects should be used unobtrusively to enhance your user's experience, not just because you can. JavaScript got a really bad reputation for introducing obtrusive animation effects that some people considered fun and cool but most people considered very annoying. With standard DOM scripting methods, you can still make the same mistakes, so use things like animation and transitions with a bit of apprehension.

Transition effects are a nice touch when you need to subtly highlight a portion of the page to indicate a change. For example, after you've completed an action in the interface, you may want to indicate if the change was a success or if it failed. Likewise, if one action has an influence on another seemingly unrelated element of the page, you may want to indicate the change there as well.

In order for a transition to work, your script needs to run a periodical method that alters the desired elements over time. For this, you'll need to use the setTimeout() JavaScript function, which invokes a method after a given number of milliseconds. By creating a number of sequential setTimeout() calls that modify the same elements with increasing time, as follows, you can create the desired transitions:

```
setTimeout(modifyElement,10);
setTimeout(modifyElement,20);
setTimeout(modifyElement,30);
setTimeout(modifyElement,40);
setTimeout(modifyElement,50);
//etc...
```

Transition effects are highly customized, so in many cases, you'll have to roll your own solutions, but in Chapter 10, we'll look at a number of JavaScript libraries, many of which have a variety of built-in transition methods to make things a little easier. Until then, let's get your creative juices flowing with a quick example of a fade between two colors:

Using the following fadeColor() method from the chapter5/fadeColor/test.html example, you can specify the two colors as well as the callback method:

```
function fadeColor( from, to, callback, duration, framesPerSecond) {

    // A function wrapper around setTimeout that calculates the
    // time to wait based on the frame number
    function doTimeout(color,frame) {
        setTimeout(function() {
            try {
                callback(color);
            } catch(e) {
                // Uncomment this to debug exceptions
                // ADS.log.write(e);
            }
        }, (duration*1000/framesPerSecond)*frame);
    }

    // The duration of the transition in seconds
    var duration = duration || 1;
    // The number of animated frames in the given duration
    var framesPerSecond = framesPerSecond || duration*15;

    var r,g,b;
    var frame = 1;

    // Set the initial start color at frame 0
    doTimeout('rgb(' + from.r + ',' + from.g + ',' + from.b + ')',0);

    // Calculate the change between the RGB values for each interval
    while (frame < framesPerSecond+1) {
        r = Math.ceil(from.r
            * ((framesPerSecond-frame)/framesPerSecond)
            + to.r * (frame/framesPerSecond));
        g = Math.ceil(from.g
            * ((framesPerSecond-frame)/framesPerSecond)
            + to.g * (frame/framesPerSecond));
        b = Math.ceil(from.b
            * ((framesPerSecond-frame)/framesPerSecond)
            + to.b * (frame/framesPerSecond));

        // Call the timeout function for this frame
        doTimeout('rgb(' + r + ',' + g + ',' + b + ')',frame);

        frame++;
    }
}
```

To invoke the transition, define two JavaScript objects with `r`, `g`, and `b` properties for the starting and ending colors and provide a callback method to apply the new transition color to whichever elements you like. The callback function will receive one argument containing the color in the format `rgb(#,#,#)`:

```
fadeColor(
     {r:0,g:255,b:0}, // Star color
     {r:255,g:255,b:255}, // End color
     function(color) {
          // Apply the color to your elements
          ADS.setStyle('element',{'background-color':color});
     }

);
```

Let’s revisit the address postal code example from Chapter 4, where entering a postal code prepopulated the rest of the address fields. It would be beneficial to indicate the successful prepopulation by highlighting the change. Using this `fadeColor()` method would provide a subtle yet informational touch to the user interface, and it would only involve adding a few lines to the postal code’s `XMLHttpRequest` `onreadystatechange` method:

```
req.onreadystatechange = function() {
     if (req.readyState == 4) {
          eval(req.responseText);

          if(ADS.$('street').value == '') {
               ADS.$('street').value = street;
               fadeColor(
                    {r:0,g:255,b:0},{r:255,g:255,b:255},
                    function(color) {
                         ADS.setStyle('street',
                             {'background-color':color}
                         );
                    }
               );
          }

          if(ADS.$('city').value == '') {
               ADS.$('city').value = city;
               fadeColor(
                    {r:0,g:255,b:0},{r:255,g:255,b:255},
                    function(color) {
                         ADS.setStyle('city',
                             {'background-color':color}
                         );
                    }
               );
          }

          if(ADS.$('province').value == '') {
               ADS.$('province').value = province;
               fadeColor(
                    {r:0,g:255,b:0},{r:255,g:255,b:255},
                    function(color) {
                         ADS.setStyle('province',
                             {'background-color':color}
                         );
                    }
               );
          }
     }
}
```

When populated, the fields will fade from green to white, indicating a successful prepopulation of the information.

Summary

Dynamically modifying the presentation of elements is a relatively simple task, but doing it right can be a challenge. In this chapter, you looked at a few of the common DOM2 Style objects that you'll be using, and you added a number of methods to your ADS library that provide browser-agnostic access to those objects.

The importance of separating CSS presentation from your DOM script was also stressed throughout the chapter. Some of the methods and techniques included modifying the `style` property and `className` switching for smaller changes, as well as style sheet switching and CSS rule modification for more global changes.

These concepts will be important to keep in mind, not only for large projects with multiple designers and developers but also for small projects that are just yours and yours alone. Separating your presentation from both the document structure and behavioral enhancements helps to future-proof your web application and will inevitably save you time, effort, and money down the road when you decide things are getting a little out of date and you need a redesign.

Now, let's look at a case study in Chapter 6; we'll take everything we've learned so far and use it to create an unobtrusive image resizing and cropping tool.