Tricking Browsers and Hiding Styles

It's a crying shame that CSS, designed to be so simple and approachable to nonprogrammers, has turned into such a cabalist's affair!

—*Todd Fahrner*

Although some feel the browser wars are behind us, their sad legacy persists. Every time an author has to code a workaround to make Navigator 4.x happy, every time a Web page comes up differently in different browsers, an echo of the wars lives on. Fortunately, there are ways to take advantage of these same browser flaws to make our lives a little easier. It might not seem like much, but we need to take our victories where we can.

Although these tricks probably seem like a lot of effort just to cater to browser bugs, they're less trouble than setting up JavaScript sniffers to serve up different style sheets to different browsers, let alone writing the separate style sheets that such an approach would require. If you can get away with ignoring old, flawed browsers, then more power to you! Write to the standards and sleep well at night. But if, like most of us, you have to worry about what your site will look like in every browser out there, one or more of these tricks might be just the thing.

Turning Browser Flaws to Our Advantage

This looks at four ways to hide CSS from certain browsers. Three of these methods are ways to hide entire style sheets from buggy browsers, but one allows authors to hide parts of a single rule. Best of all, these methods are all perfectly valid CSS and don't require any form of scripting to carry out. I think of them as "CSS aikido."

Using `@import` Restrictions

Old Explorers

`@import` is also not understood by IE3/Win and IE4/Win, if that's a concern for you.

If you've ever struggled with Netscape Navigator 4.x, this trick is for you. It all hinges on the fact that NN4.x doesn't understand what `@import` means. This is by far the most widely used trick, even though it's only good for hiding external style sheets.

Consider the following style sheet:

```
<style type="text/css">
h1 {border-bottom: 2px solid gray; margin-bottom: 0.25em;}
a:link, a:visited {padding: 1px 0.5em; background: #FF9;
   font-weight: bold;}
a:link {color: blue;}
a:visited {color: purple;}
a:hover {color: red; background: yellow;}
a:active {color: yellow; background: red;}
</style>
```

A few things here will cause trouble or otherwise be useless in NN4.x. Most obviously, NN4.x doesn't understand `a:hover` rules because it was released before `:hover` made its way into the CSS specification. It also has trouble with `:active`. These aren't necessarily major problems because NN4.x will pretty much ignore these rules. There is a much bigger problem, unfortunately: The application of `padding` to hyperlinks.

For some reason, NN4.x seems to assume that any element that's been given padding (or borders or margins) must be a block-level element. Thus, the styles set up here will cause hyperlinks to break up whatever element they're found within and possibly overlap other pieces of the document. So you definitely can't have NN4.x seeing the `padding` declaration in your CSS.

Here's the solution: Take the Navigator-unfriendly styles and move them into a separate file (call it `adv-styles.css`, for example). That would give you:

```
a:link, a:visited {padding: 1px 0.5em;}
a:hover {color: red; background: yellow;}
a:active {color: yellow; background: red;}
```

This leaves you with a somewhat pared-down style sheet:

```
<style type="text/css">
h1 {border-bottom: 2px solid gray; margin-bottom: 0.25em;}
a:link, a:visited {background: #FF9; font-weight: bold;}
a:link {color: blue;}
a:visited {color: purple;}
</style>
```

Now all you have to do is import the styles you just removed.

```
<style type="text/css">
@import url(http://www.example.org/styles/adv-style.css);
h1 {border-bottom: 2px solid gray; margin-bottom: 0.25em;}
a:link, a:visited {background: #FF9; font-weight: bold;}
a:link {color: blue;}
a:visited {color: purple;}
</style>
```

As far is NN4.x is concerned, the `@import` line might as well not be there, so it will never see the styles contained in the imported style sheet. Pretty much any other browser will import and use those styles, so they get the benefit of the more advanced CSS.

A variation on this trick is to make all of the styles external and then use the `link` element to bring in the Navigator-friendly styles. (Even though NN4.x doesn't understand `@import`, it does understand `link`.)

```
<link rel="stylesheet" type="text/css" href="basic-styles.css">
<style type="text/css">
@import url(adv-styles.css);
</style>
```

This works in concert with the style sheets shown in Listings 1 and 2.

Listing 1 The Contents of `basic-styles.css`

```
h1 {border-bottom: 2px solid gray; margin-bottom: 0.25em;}
a:link, a:visited {background: #FF9; font-weight: bold;}
a:link {color: blue;}
a:visited {color: purple;}
```

Listing 2 The Contents of `adv-styles.css`

```
a:link, a:visited {padding: 1px 0.5em;}
a:hover {color: red; background: yellow;}
a:active {color: yellow; background: red;}
```

Which One Is Better?

Although both approaches are worthwhile, it's usually easier to use the `link/@import` method instead of putting an `@import` into an embedded style sheet. The `link/@import` trick has gained a fair amount of popularity on the Web, so it's well understood and there are plenty of people who can help out if any trouble arises.

Media Attributes

It's also possible to hide `link`ed and embedded style sheets from Navigator 4.x simply by making the styles available in all media. For example:

```
<link rel="stylesheet" type="text/css" href="advanced-styles.css"
   media="all">
<style type="text/css" media="all">
h1 {border-bottom: 2px solid gray; margin-bottom: 0.25em;}
a:link, a:visited {background: #FF9; font-weight: bold;}
a:link {color: blue;}
a:visited {color: purple;}
</style>
```

Media-value Pitfalls

This trick only works on NN4.x, and it has been reported occasionally that, in rare cases, it can crash NN4.x on some machines.

Both the `link` element and the entire embedded style sheet will be ignored by NN4.x because it doesn't understand any value other than `screen`. You can also fool NN4.x by using combinations of media:

```
<link rel="stylesheet" type="text/css" href="advanced-styles.css"
   media="screen, print">
<style type="text/css" media="screen, projection">
h1 {border-bottom: 2px solid gray; margin-bottom: 0.25em;}
a:link, a:visited {background: #FF9; font-weight: bold;}
a:link {color: blue;}
a:visited {color: purple;}
</style>
```

Any style sheet (or `link`) you mark as applying only to `screen` media will be seen and used by NN4.x, so you might have to use other tricks (such as `@import`) to hide advanced styles from NN4.x.

The Highpass Filter

With this trick, we move out of the relatively straightforward tricks we've been examining and into some seriously deep work. At first blush, this trick (and the next one) look like gross hacks—and in a sense they are—but they're also fully valid CSS and an elegant approach to the subject of hiding styles from browsers. Because of their clever use of obscure bugs in CSS parsers, I sometimes refer to them as "parsermancy."

Credit Where It's Due

Both the highpass filter and the `voice-family` trick (see the next section) were developed and published by Tantek Çelik, and they are included here with his kind permission. You can find these and more by visiting `http://www.tantek.com/` and clicking on "CSS Examples."

Let's just look at the code involved and then work out what's happening.

```
@import "null?\"\{";
@import "adv-styles.css";
```

The second line looks normal enough, but the first one is seriously warped. You might be wondering who let the cat walk on the keyboard or if CSS is starting to become Perl, but neither is the case.

This trick relies on the poor CSS parsing that is present in older browsers, primarily the Internet Explorer for Windows line up through IE5.x. Due to bugs in their CSS parsing engines, IE3.x through IE5.x think the preceding code is:

```
@import null?\
{ ";@import" adv-styles . css ";
```

Neither line leads to any useful results, so the effect is to hide the external style sheet `adv-styles.css` from these browsers.

In browsers with properly written CSS parsers, the rules are resolved like this:

```
@import "null?"{";
@import "adv-styles.css";
```

The first line is harmlessly parsed and handled. Because it points to nothing, it has no effect on the styling of the document. The second line is a normal `@import`, so the style sheet to which it points will be applied to the document.

Listings 3 and 4 show simple test documents you can use to see if a given browser can handle the code properly or if its parser has flaws. Make sure the files are in the same directory when you test. If the text is styled as described, the browser passes the test; otherwise, it fails.

Listing 3 A Basic Test Document

```
<html><head><title>High Pass Filter Test</title>
<style type="text/css">
@import "null?\"\{";
@import "filtertest.css";
</style>
</head>
<body>
<p id="test">This text should be green and boldfaced
  with a yellow background.</p>
</body>
</html>
```

Listing 4 The Contents of `filtertest.css`

```
p#test {color: green; font-weight: bold; background: yellow;}
```

Among the browsers that will not be fooled by the highpass filter are IE5/Mac, IE6/Win (when in strict mode), Opera 5, Netscape 6.x, and OmniWeb for Macintosh.

The `voice-family` Hack

The second piece of parsermancy, the `voice-family` hack, shares some things in common with the highpass filter trick. In fact, the highpass filter is based on the

Voice/Box

This trick is better known as the "box model hack" because that's what Tantek calls it. As you'll soon see, however, this trick is good for much more than helping browsers with broken box models, so it's been renamed here.

`voice-family` hack. The difference is that whereas the highpass filter hides an `@import` statement, the `voice-family` hack hides just a portion of a rule. The size of that portion is entirely up to you.

Again, let's see the trick in action and then delve into how it works. Listing 5 shows an entire test file.

Listing 5 Testing the `voice-family` Trick

```
<html><head><title>voice-family trick</title>
<style type="text/css">
div#test {
  color: red;
  voice-family: "\"}\"";
  voice-family:inherit;
  color: green;
}
</style>
</head>
<body>
<div id="test">This DIV will be green in browsers that parse correctly
and red in those that don't.</div>
</body>
</html>
```

Hooray, more random-looking characters! Just like the highpass filter, this trick relies on the poor CSS parsing that is present in older browsers, primarily the Internet Explorer for Windows line up through IE5.x. Here's what a correctly implemented parser sees, with a little extra whitespace for clarity:

```
div#test {
  color: red;
  voice-family: "}" ;
  voice-family:inherit;
  color: green;
}
```

That's right, the browser is looking for a `voice-family` file called `"}"`. If such a file existed on the user's computer, an audio browser could use it as the voice with which to read the page. Of course, no Web browsers are able to speak Web pages, so this rule would have no effect. The second `voice-family` rule is there to clean up after the first, just in case an audio browser reads the styles and a file with that name exists on the users' computer. The second rule causes the first to be ignored even by audio browsers.

However, here's what an older browser with a flawed parser will see:

```
div#test {
  color: red;
  voice-family: "\"}
  \"";
  voice-family:inherit;
  color: green;
}
```

In other words, as far as these flawed browsers are concerned, the `div#test` rule ends with the first `voice-family` declaration. The rest of the rule is split off and effectively ignored.

You might be wondering what good this does you. Consider the differences in element box sizing, as explained in "Picking a Rendering Mode" (which is available on this Web site) and touched on in Project 9, "Multicolumn Layout." It's possible to write rules that include `width` values for both old and new browsers so that consistent element sizing becomes a reality. For example, let's say you want your test `div` to be 400 pixels wide, as measured from outer-left border edge to outer-right border edge.

```
div#test {
  padding: 20px;
  border: 5px solid gray;
  width: 400px;
  voice-family: "\"}\"";
  voice-family:inherit;
  width: 350px;
}
```

Browsers like Explorer 5.5, which takes `width` to be the distance from border edge to border edge, will think this to be the rule for `div#test`:

```
div#test {
  padding: 20px;
  border: 5px solid gray;
  width: 400px;
  voice-family: "\"}
```

Properly written browsers, on the other hand, will see the entire rule as you wrote it and will use the declaration `width: 350px` because it comes later in the rule. Thus, the test `div` will be sized the same in both old and new browsers despite the differences in element sizing. (The same would be true for `height` declarations as well, if you want to make them.)

There is one potential stumbling block, and that's Opera. Although Opera follows CSS with regard to element sizing, it suffers from some of the same parser flaws as IE5.x for Windows. Thus, it will only see the `width: 400px` declaration but will still use that value for the width of the content-area of the element. This is where the be-nice-to-Opera rule comes in:

```
html>body div#test {width: 350px;}
```

Translating the Selector

The CSS2 selector `html>body div#test` is read "any `div` element with an `id` whose value equals `test` that is a descendant of a `body` element that is a child of an `html` element." You can get a translation of any valid CSS selector from the SelectORacle (`http://gallery.theopalgroup.com/selectoracle/`).

This rule uses CSS2 selectors that IE5.x (and earlier) doesn't understand but Opera does. Thus, Opera gets the value it needs after all. Actually, so will some other well-written browsers, but because it's the same value as they got from the previous rule, there's no harm done. The entire set of code ends up as shown in Listing 6.

Listing 6 Working Around Various Parser Flaws

```
div#test {
padding: 20px;
  border: 5px solid gray;
  width: 400px;
  voice-family: "\"}\"";
  voice-family:inherit;
  width: 350px;
}
html>body div#test {width: 350px;}
```

Although this trick is most often used to make element sizes consistent, it could be used for any number of purposes. One such example is based on the varying default values for `font-size`. Explorer 4.x and 5.x for Windows assume the default value for `font-size` to be `small`. Netscape 4.x and 6.x follow the CSS specification and use the default value `medium`. This has led most authors to avoid these keywords because declaring `font-size: medium` would actually make the text larger than the user's default in IE5.x for Windows!

Thanks to the `voice-family` hack, you can get around this inconsistency. Let's say you actually want the text to be one step above the default font size. That should be `large`, but as previously mentioned, IE4.x/Win and IE5.x/Win think that `medium` is one step above the default. Thus:

```
div#test {
  font-size: medium;
  voice-family: "\"}\"";
  voice-family:inherit;
  font-size: large;
}
```

Such an approach can deliver, for the first time in years, a more consistent font sizing when using the `font-size` keywords.

In general, the `voice-family` hack can be used to hide any number of declarations from IE4.x, IE5.x, and Opera. For example, you might want to avoid the `background-attachment` bugs in these browsers.

```
div {
  background: aqua url(bg.jpg) center no-repeat scroll;
  voice-family: "\"}\"";
  voice-family:inherit;
  background-attachment: fixed;
}
```

For that matter, you could hide an entire declaration block from flawed browsers by starting it with the `voice-family` hack.

```
div {
  voice-family: "\"}\"";
  voice-family:inherit;
  background: aqua url(bg.jpg) center no-repeat fixed;
  color: green;
}
```

This would keep the particular green-on-aqua-with-background-image combination from older browsers. The advantage is that this only hides the one rule from flawed browsers without disrupting the rest of the style sheet.