CHAPTER 13

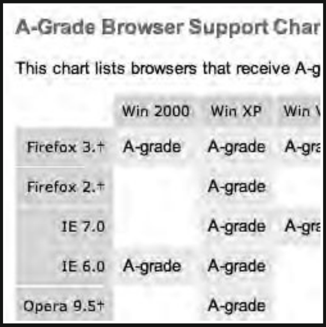
HARNESSING THE POWER OF EXTERNAL STYLE SHEETS

Google Famous Prima

Web

Famous Monkeys Through Histor
Also, when encountering a monkey she d
her disapproval of the other **primate**. ...
www.ape-o-naut.org/**famous**/**famous**/real

Famous Monkeys Through I
Noam Chomsky: They have done i
primates. But it has about as muc
www.ape-o-naut.org/**famous**/ - 1k -


A-Grade Browser Support Char

This chart lists browsers that receive A-g

	Win 2000	Win XP	Win \
Firefox 3.†	A-grade	A-grade	A-gra
Firefox 2.†		A-grade	
IE 7.0		A-grade	A-gra
IE 6.0	A-grade	A-grade	
Opera 9.5†		A-grade	

We're almost there! This is our last practical, hands-on chapter, and we'll use it to introduce some quite advanced topics; but if you've been a diligent Web Standardista and done your homework, we're confident you're ready.

At this point we have a well-structured and well-presented King Kong page; however, there are still a number of fundamental ways in which we could improve it. Not least by introducing you to the power of using external style sheets, where the real benefit of a web standards–based approach becomes clear.

We'll use the contents of a typical `head` element to form the basis of the majority of this chapter's journey, introducing a number of useful new tags and elements into the `head` that allow you to really get the most out of the Web Standardistas' approach. In particular we'll cover the use of `<meta>` tags, looking at what they're useful for and exposing a few myths about them along the way. These are the tags and elements that will really set you apart from your less well-trained peers.

The primary focus of the chapter will be to look at the benefits of using external style sheets: taking the style sheet we've created for our King Kong page, removing it from the King Kong page itself, and linking to it as a separate file. Doing this will allow us to link *all* of our ape and monkey web pages to the same style sheet, allowing us to focus our efforts on a single style sheet that styles *all* of the Famous Primates web site's pages, improving efficiency and ensuring consistency across the site. This is where the real power of CSS lies. We'll also show you how to create a separate print style sheet so your web pages look fantastic in print too.

Lastly, we'll cover testing and troubleshooting, two aspects essential to the Web Standardistas' approach. Without further ado, let's get started.

The head elements that make it all happen

According to the W3C's recommended standard, only a few elements are legal inside the `head` element: `base`, `link`, `meta`, `title`, `style`, and `script`. You've encountered some of these elements before, not least the `meta`, `title`, and `style` elements with which you're now well and truly acquainted, but the remainder are probably new to you.

Now that we're getting toward the end of the book and this kind of detail is no longer intimidating to you, we're going to take a look at a typical `head` element that you might encounter on a web page "in the wild." We'll break it apart and look at what each element is doing. This will introduce some important new concepts that will form the closing part of our Web Standardistas journey.

One element we're not going to focus on is the `title` element; you're well acquainted by now and we don't need to reveal any more of its secrets to you. We will, however, look at the `style` element where, up until this point, we've been locating our CSS.

We'll examine the `style` element in relation to the `link` element, showing how you can use it to offload your CSS to an *external* style sheet so that all of your documents are linked to one, easily updated style sheet stored in a central location. This is where the real power of a combined XHTML and CSS approach lies: using a *single* CSS file to style *all* of your web pages.

Once we've covered that, we'll run through the remainder of the elements you'll find in the head section, introducing you to each, one by one. This will give the chapter some structure around which we can build, as we introduce each new element and its purpose.

The following example shows some markup that you might expect to find on a typical web page. We've expanded upon our King Kong page to tie it back into the process we've been working through for the last few chapters. Many of these elements are new; however, we'll walk through them one by one and introduce you to them.

```
<!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>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>King Kong | Apes in the Movies | Famous Primates</title>
  <meta name="description" content="King Kong is a fictional giant ape
  from the legendary Skull Island. A renowned ape thespian he is
  unquestionably a famous primate." />
  <meta name="author" content="Web Standardistas" />
  <link rel="stylesheet" type="text/css" href="../css/screen.css"➥
  media="screen" />
  <link rel="stylesheet" type="text/css" href="../css/print.css"➥
  media="print" />
  <!--[if lte IE6]>
  <link rel="stylesheet" type="text/css" href="../css/iehacks.css" />
  <![endif]-->
  <link rel="shortcut icon" type="image/ico" href="../favicon.ico" />
  <script type="text/javascript" src="../js/primates.js"></script>
</head>
```

Clearly the preceding is a little more complicated than the web pages we've been creating up until this point. At first glance all these extra elements in the head might appear a little intimidating; however, as we introduce each of them to you, you'll become familiar with them in no time.

The importance of meta tags

The first element we're going to introduce properly is the `<meta>` tag and its attributes. You first met a `<meta>` tag in Chapter 2 when we introduced character encoding, and if you've been using the template file we provided for you, you should be used to seeing it by now. In this section we'll formally introduce you, revealing its secrets. `<meta>` tags are used for much more than character encoding, however; let's take a look at them in a little more detail now.

In computing terms, **metadata** is data that describes other data and, as you'll see, one use of the `<meta>` tag is to describe the contents of a web page. In this section we'll look at some uses of the `<meta>` tag and its various attributes in action.

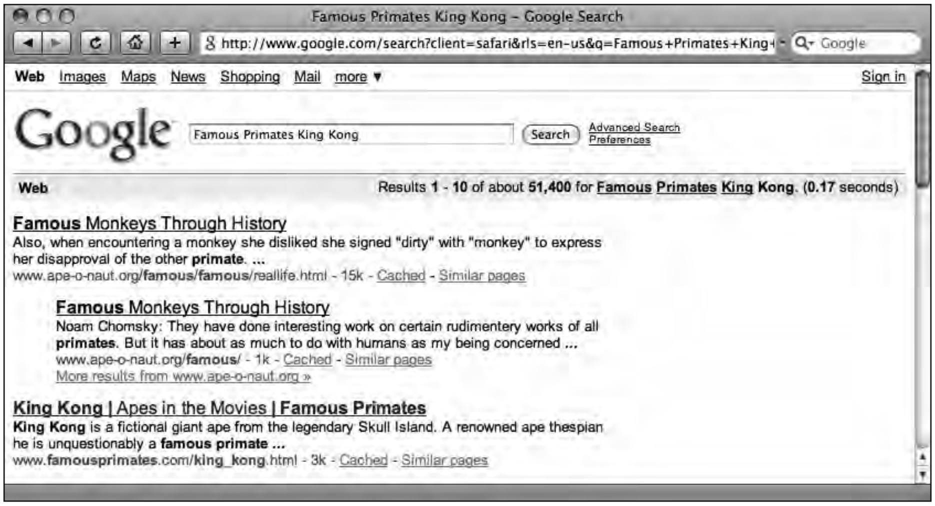
To save you constantly referring back to our introductory example, we'll include the relevant lines of code from the head element as we introduce each section. In this section we'll be looking at the `<meta>` tags as follows:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
...
<meta name="description" content="King Kong is a fictional giant ape
from the legendary Skull Island. A renowned ape thespian he is
unquestionably a famous primate." />
<meta name="author" content="Web Standardistas" />
```

You might have heard of `<meta>` tags in discussions about Google and search engine optimization that possibly express the importance of "using keywords in meta tags" and the benefits of using `<meta>` tags to provide a description for search engine purposes.

In fact Google relies less on `<meta>` tags for keywords now, looking instead at the actual content of a page to establish search rankings. This is largely due to the manipulation of keywords, where less-than-scrupulous search engine optimization (SEO) consultants would fill the `keywords` attribute with content often of no relevance to the page in an effort to boost search engine rankings (a tactic known as keyword stuffing). As search engine robots—the programs that index your web pages—have become more sophisticated, the value of keywords in `<meta>` tags has declined considerably.

Although `meta keyword` tags are no longer really relevant, `meta description` tags are still relevant and can be used to provide the description that the search engine will use in its listing of your site as shown in Figure 13-1.


Famous Primates King Kong – Google Search
http://www.google.com/search?client=safari&rls=en-us&q=Famous+Primates+King+
Google
Web Images Maps News Shopping Mail more ▾
Sign in
Famous Primates King Kong
Search
Advanced Search
Preferences

Web Results 1 - 10 of about 51,400 for Famous Primates King Kong. (0.17 seconds)

Famous Monkeys Through History
Also, when encountering a monkey she disliked she signed "dirty" with "monkey" to express her disapproval of the other primate. ...
www.ape-o-naut.org/famous/famous/reallife.html - 15k - Cached - Similar pages

Famous Monkeys Through History
Noam Chomsky: They have done interesting work on certain rudimentery works of all primates. But it has about as much to do with humans as my being concerned ...
www.ape-o-naut.org/famous/ - 1k - Cached - Similar pages
More results from www.ape-o-naut.org »

King Kong | Apes in the Movies | Famous Primates
King Kong is a fictional giant ape from the legendary Skull Island. A renowned ape thespian he is unquestionably a famous primate ...
www.famousprimates.com/king_kong.html - 3k - Cached - Similar pages

Figure 13-1. The King Kong page's `meta description` attribute displaying as a description of the page in Google

<meta> tags aren't just for search engines, however, as the example of a typical head element earlier in the chapter showed. We've included two other <meta> tags; the first contains a character encoding that we introduced in Chapter 2, but will explain in detail in a moment:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

The second specifies the authors of the specific page:

```
<meta name="author" content="Web Standardistas" />
```

As the three examples in this section suggest, <meta> tags can be created for almost anything. There are a number of initiatives, not the least the Dublin Core Metadata Initiative (DCMI), that are developing metadata standards. The DCMI (www.dublincore.org), established in 1995, is dedicated to creating a simple and standardized set of metadata conventions for describing documents online in ways that make them easier to find. Microformats (www.microformats.org), a more recently emerging set of markup principles, is also aimed at exploring metadata and information markup.

It's all in a name

<meta> tags have four types of attribute: http-equiv, name, content, and scheme. You met the name attribute earlier, when we introduced the description attribute. In fact there are a number of potential values for the name attribute, one of which is included in our typical head element example:

```
<meta name="author" content="Web Standardistas" />
```

No prizes for guessing that this is used to identify the authors of the particular page (in this case the authors of the book you're now holding). The following are a few other name attributes:

- keywords: As we already mentioned earlier, Google and other search engines no longer place any emphasis on keywords in <meta> tags. You should instead ensure your web page's content has a rich mix of keywords marked up within the h1, h2, and p elements on the page itself.
- copyright: This attribute is useful for including copyright declarations and information.
- robots: This attribute is useful for informing search engines which pages you would like to be indexed (and equally importantly, which pages you wouldn't).

There are a number of other potential values for the name attribute. We've covered these and the topic of the <meta> tag and its attributes in greater depth at the book's companion web site:

```
www.webstandardistas.com/periodical
```

Speaking a foreign language

You first met the http-equiv attribute in Chapter 2 when we briefly introduced you to character encodings. The following <meta> tag provides information to the browser about how to interpret the contents of our XHTML page. Let's take a look at it more closely now to see how it works its magic:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

The parts we're interested in are the values in the content attribute: the first value, text/html, informs the browser that the document should be treated as text/html, and the second instructs the browser to use a character set (charset) of UTF-8. Lost yet? Resist the urge to skip this section; we encourage you to stick with us to the end as the UTF-8 character set is important.

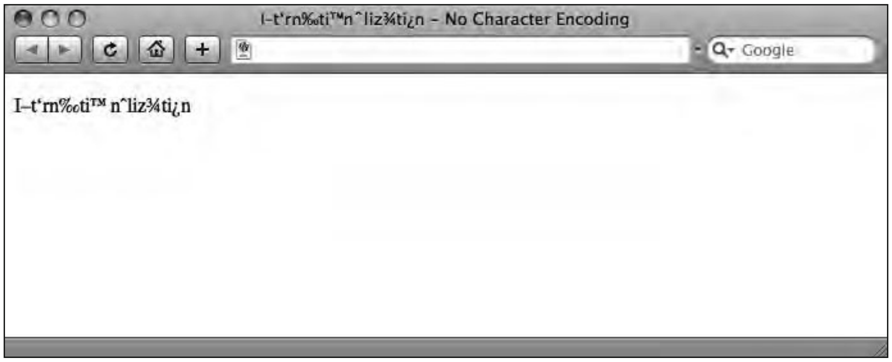
UTF-8 (which stands for 8-bit Unicode Transformation Format) supports a large character set allowing you to, for example, include words with umlauts and accents on your web pages.

The easiest way to explain the benefits of using UTF-8 is to show two examples in action: one web page with UTF-8 specified as a character set and one without. We'll use an invented but very foreign-looking word—Iñtërnâtiônàlizætiøn—that uses a variety of unusual characters in two different web pages and look at how the character encoding, or lack of, affects the display of the characters in the browser.

The following very short page has no character encoding specified:

```
<!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>Iñtërnâtiônàlizætiøn - No Character Encoding</title>
  </head>
  <body>
    <p>Iñtërnâtiônàlizætiøn</p>
  </body>
</html>
```

This page displays in the browser as shown in Figure 13-2. Note the page's title and the p within the body. Disaster. With no character encoding specified, the web page doesn't display what we'd like it to. Clearly we need to fix this.



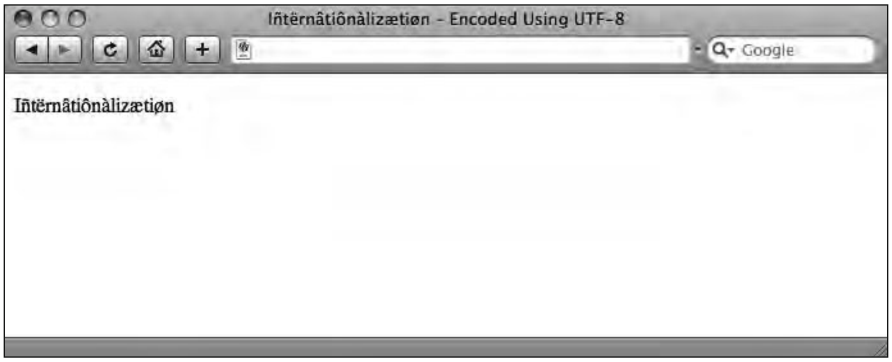
I–t'rn%‰ti™ n^liz¾ti¿n

Figure 13-2. Our web page with no character encoding struggles with obscure characters.

Now take a look at the same page, with our character encoding specified as UTF-8:

```
<!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>
    <meta http-equiv="Content-Type" content="text/html; ➥
    charset=UTF-8" />
    <title>Iñtërnâtiônàlizætiøn - Encoded Using UTF-8</title>
  </head>
  <body>
    <p>Iñtërnâtiônàlizætiøn</p>
  </body>
</html>
```

This page displays in the browser as shown in Figure 13-3—perfect. By specifying UTF-8 as a character set, our web page can now support a wide variety of characters, allowing us to create pages that are internationally friendly.



Iñtërnâtiônàlizætiøn

Figure 13-3. Our web page with UTF-8 character encoding specified works fine.

13

Character encoding is a tricky topic, and we've barely scratched the surface here. We'd strongly suggest that anyone involved in working with an international character set reads up further on this. To help with this, we've provided additional resources at the book's companion web site:

```
www.webstandardistas.com/resources
```

External Style Sheets

Next on our quest to unravel the head is the `link` element. As we briefly alluded to earlier, we'll use the `link` element to enable us to offload our style sheets and link to them. Up until this point all of our style sheets have been written on the pages we've been working on and embedded within them; now we'll separate our XHTML pages (the content) and CSS files (the presentation).

This section is where the real power of CSS comes into play in full effect.

Although it's useful during the development phase to work with all the XHTML and CSS in one file, we've now reached a point with our King Kong page that it's nearly finished. This is the stage at which we take all the information we've developed in the embedded style sheet and offload it to an external style sheet, creating a single, external style sheet that all of our documents link to.

Before we show you the relevant markup, we'll pause for a moment and consider the pros and cons of embedded style sheets as opposed to linked style sheets.

Embedded vs. linked style sheets

As we've mentioned, throughout the previous chapters we've used an embedded style sheet to develop our King Kong web page. The benefit of using an embedded style sheet *during the development phase* is that it allows you to tightly control and test your web page as it evolves from within one easy location.

However, there are a number of downsides to using embedded style sheets. Unless every single page on your web site is different—an unlikely event—linking to a single CSS file is a better approach for a number of reasons.

First, by linking all of the web pages in your web site to a single CSS file, your users' browser will only need to download the CSS file once, after which it will be accessed from the browser's cache. This speeds up download times considerably and reduces bandwidth requirements. With embedded CSS every time a page is downloaded, the CSS is downloaded, an inefficient and bandwidth-intensive process. Not only will a page with an embedded CSS file take longer to download, it will also place a heavier load on the server.

Another reason to link all your files to a single CSS file to control presentation is to improve consistency and reduce the number of files that need to be updated. Imagine every single page in your web site has the same embedded CSS. If you need to make just one change, you'll need to update all of the web pages. Clearly a single, linked, external style sheet is a better approach.

Linking to an external style sheet

The relevant part of our head element that handles the link to our external style sheet is the following line:

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

Let's take a look at this in detail and work out what's going on. The first attribute, `rel`, defines the relationship of the linked document with the current document, in this case highlighting that the link is to a `stylesheet`. The second attribute, `type`, specifies the style sheet language: `text/css`. The `href` attribute, which you're familiar with, specifies a URL pinpointing where the style sheet can be found and what it's called, in this case a file named `screen.css` in a folder called `css`. Lastly, the `media` attribute specifies the intended rendering medium or media, in this case `screen`.

In the preceding example we've set the `media` attribute to `screen`, but there are a number of other media attributes we can use.

Media types

CSS supports a wide variety of media types, although support for a number of them is inconsistent. The following media types are supported as specified by the W3C's CSS 2.1 specification (`http://www.w3.org/TR/CSS2/media.html`):

- `all`: Suitable for all devices
- `aural`: Intended for speech synthesizers
- `braille`: Intended for Braille tactile feedback devices
- `embossed`: Intended for paged Braille printers
- `handheld`: Intended for handheld devices
- `print`: Intended for printed material and for documents viewed on screen in print preview mode
- `projection`: Intended for projected presentations, for example, using data projectors
- `screen`: Intended primarily for color computer screens
- `tty`: Intended for media using a fixed-pitch character grid, such as teletypes, terminals, or portable devices with limited display capabilities
- `tv`: Intended for television-type devices

In this chapter we'll be focusing on `screen` and `print` styles, the two you are likely to find yourself using most.

It's worth noting that many of these media types are not supported by any devices at all and possibly may never be, for example, `tty` *and both Braille types (which is perhaps no bad thing given the specialist knowledge that would be required to create effective Braille style sheets).*

Using @import

An alternative method to using the `link` element to link to our style sheet is to use an `@import` rule instead as in the following example:

```
<style type="text/css">
@import url(../css/screen.css) screen;
</style>
```

Essentially what this does is inform the browser to look for a CSS file called `style.css`, import it, and use it to style the document. This was a popular method of importing styles as support for web standards evolved.

The original motivation for the `@import` rule was to hide CSS from old browsers that didn't understand CSS very well, most notably Netscape 4 (a browser you're highly unlikely to encounter in this day and age). The idea of the `@import` rule was to serve a simple style sheet or no style sheet at all to CSS-challenged browsers, while sending a full style sheet to standards-aware browsers. However, since these browsers are over a decade old, supporting them should become less and less of a concern.

One area where the `@import` rule has an advantage over the `link` element is that its use is not restricted to the `head` element of a web page; it can also be used within an external style sheet.

This allows us, for example, to link to a separate CSS file that contains a number of `@import` rules pulling in other, additional style sheets. When dealing with multiple style sheets, this approach enables you to easily manage importing external style sheets, while keeping the `head` of your XHTML files neat and tidy. We cover this in greater depth at the book's companion web site:

```
www.webstandardistas.com/periodical
```

In this day and age, whether to use `@import` or the `link` element to link to your style sheets is largely a matter of taste; both methods work, and all modern browsers understand either.

Creating our external CSS file

The creation of our external CSS file is simple and should take only a few moments. We create a new plain text document in our plain text editor and save it as `screen.css`. The `.css` extension, like the `.html` extension, identifies the file type to the browser, in this case indicating it is a Cascading Style Sheet.

We open our King Kong web page and select everything between the opening `<style type="text/css">` and the closing `</style>` tags (but not the `<style>` tags themselves). We simply copy all of our CSS rules (and any relevant comments) and paste them into the `screen.css` document we just created and save it.

Like the `images` folder we created in Chapter 6 to organize our images, it's a good idea to create a `css` folder to store your style sheets in. Although at this point you only have one style sheet, later in the chapter we'll create a `print.css` style sheet to style printed pages. Storing these separate style sheets in one location will make the management of our web site easier as it grows.

The next stage is to create a link from all the XHTML pages we'd like to style with this style sheet using the `href` attribute. This process is identical to creating any other link (to an image for example). Bear in mind the path to the CSS file will be relative to the different web pages we're linking from. Our example in the typical `head` element follows:

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

In this example our `screen.css` file is located within a folder called `css` sitting in a folder above the file we're linking from.

That's it. Now we can delete any internal style sheets from any XHTML documents we've linked to this external style sheet; the `link` element will now take care of the link to the CSS, styling the pages. One file, styling everything. Once we've deleted our `<style>` tags we load our page in a browser and test it. The result is shown in Figure 13-4.

One thing worth noting is that when you transfer your `screen.css` file into your `css` folder, you'll need to check and amend the relationship of any images you have used in your style sheet, ensuring their relative links are fixed to take into account the relative locations of the style sheet and the images specified.

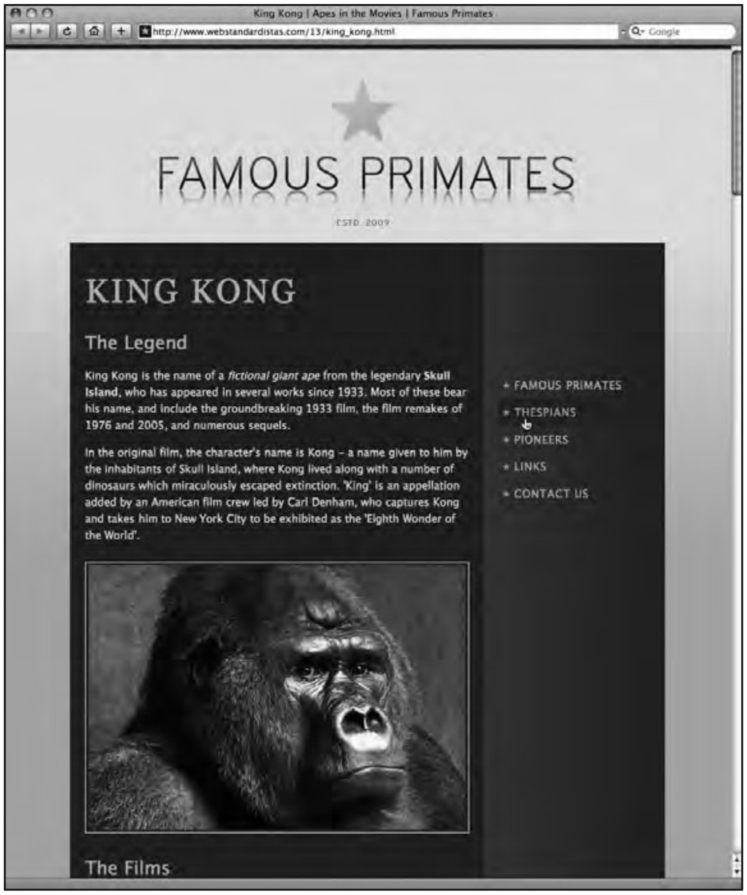
For our Famous Primates home page we've specified three background images in our style sheet. If we take one as an example, and see the effect of moving the `screen.css` file into the `css` folder, it should help you understand the principle. We take the original rule, as follows:

```
background-image: url(images/body_bg.png);
```

and change it to the following:

```
background-image: url(../images/body_bg.png);
```

This step is essentially the same as the one you took in Chapter 6 when you fixed the links to your images after putting them into the `images` folder.


King Kong | Apes in the Movies | Famous Primates
http://www.webstandardistas.com/13/king_kong.html
Google
FAMOUS PRIMATES
ESTD. 2009

KING KONG

The Legend

King Kong is the name of a *fictional giant ape* from the legendary **Skull Island**, who has appeared in several works since 1933. Most of these bear his name, and include the groundbreaking 1933 film, the film remakes of 1976 and 2005, and numerous sequels.

In the original film, the character's name is Kong – a name given to him by the inhabitants of Skull Island, where Kong lived along with a number of dinosaurs which miraculously escaped extinction. 'King' is an appellation added by an American film crew led by Carl Denham, who captures Kong and takes him to New York City to be exhibited as the 'Eighth Wonder of the World'.

The Films

- FAMOUS PRIMATES
- THESPIANS
- PIONEERS
- LINKS
- CONTACT US

Figure 13-4. Our King Kong page with the linked style sheet displays exactly the same as it did when the style sheet was embedded.

The real power of CSS

By moving all of our styles to an external style sheet and linking *all* of our Famous Primates web pages *to this one file*, we can style everything from one central location. This is where the real power of CSS lies.

Using linked style sheets will save you a significant amount of time in the long run and considerably cut down the amount of time and effort it takes to make changes to the presentational aspects of your web site. **Change the one CSS file, and the entire web site is updated automatically.** Good times!

But it gets better; we can build multiple style sheets for different purposes, all styling the same well-structured markup. For example, we might build a print style sheet for printed pages or a high-contrast style sheet with larger type for visually impaired users. The flexibility is the key.

In the next section we demonstrate this by building a print style sheet, showing how we can completely alter the look and feel of the King Kong page just by changing the style sheet.

Adding a print style sheet

One of the beauties of CSS lies in its flexibility. Having built a solid foundation of well-structured semantic markup, we can now quickly and easily restyle the very same markup to create a print style sheet that is tailored to the world of the printed page.

Clearly, printed pages have different characteristics from pages viewed onscreen. Screen and print are two different media with two distinctive sets of requirements. It makes sense to provide a print style sheet for your users that is optimized for the medium of print. Some key aspects to consider when designing a print style sheet include the following:

- Hide elements that aren't useful in print. It makes sense to hide certain aspects of your markup when developing the print style sheet, for example, navigation (after all, a user can't click links on a printed page!). CSS allows you to use `display: none;` to switch off the elements you'd prefer not to show in print, perfect for this purpose.
- Create a print-specific logo. If you've used subtle gradients for onscreen purposes that might not print well, include a version of your logo and hide it in the screen style sheet, but show it in your print style sheet.
- Ensure your print style sheet works in black and white and high contrast. When setting white text on a dark background color, bear in mind that background images and colors do not print by default. Some browsers will print just the text—*in white*—and no background; the resulting printed page will be a challenge to read!
- Ensure the sizes you specify work in print. A `#container` set at 760 pixels for screen is better set in inches (or centimeters, for countries that have embraced the metric system). Consider the width of the printed page and design accordingly.
- Use points for font sizes. In print—as with any common word processors—you'll notice type is set in points, usually 10 pt for legible body copy. When designing your print style sheet, use sizes geared toward the world of the printed page for the best results.

Building the print style sheet

Let's get started. In our example of a typical head element earlier, we have the following `link` element, specifying a link to a print style sheet:

```
<link rel="stylesheet" href="../css/print.css" type="text/css" ➥
media="print" />
```

We'll now walk you through the process of creating this style sheet. The first step in the process is to open a new file in our plain text editor and save it as `print.css`; we'll save it in the `css` folder to keep all of our CSS files organized.

Style the body

We add our first rule as follows, styling the body so that it's perfect for print:

```
body
{
font-family: Georgia, serif;
font-size: 10pt;
color: black;
background: white;
width: auto;
margin: 0 10%;
}
```

By now you should know enough about CSS to have a clear understanding of what's happening here. We've specified Georgia, a serif font, for our print style sheet. We could have used Lucida Grande as we specified in our screen.css style sheet; however, serif fonts are generally considered easier on the eye on paper (and Georgia is a beautifully designed typeface).

You'll notice we've set the font-size to 10pt, using points as a unit of measure instead of pixels. Pixels are perfect for screen; points are perfect for paper. We've set the page to print black on white, not the dark background-color of our screen style sheet; think how much toner that would eat up!

Lastly, we set the width of the page to auto, filling the page to use less paper, and set a margin of 10% on the left and right, adding a little space around our content.

Hide unnecessary content

Onscreen and in a browser our list of links in the sidebar is perfect, enabling us to navigate throughout the site. In print, links aren't clickable, so printing them is of no benefit to the user. We hide the sidebar using the display property of CSS, setting the display to none as follows:

```
#sidebar
{
display: none;
}
```

Style the headings

The next stage is to style the headings and add a little padding to space out our paragraphs a little to aid our printed page's legibility. We do this by adding the following rules:

```
h1
{
font-size: 18pt;
}

h2
{
```

```
font-size: 16pt;
}

h2
{
font-size: 14pt;
}

h4
{
font-size: 10pt;
text-transform: uppercase;
letter-spacing: 0.4em;
margin-top: 10mm;
padding-top: 5mm;
border-top: solid 1px black;
}

p
{
margin: 0 0 5pt 0;
}
```

Style the links

Although we switched off the links in our sidebar by using display: none;, we'd like to display our links in our references section and the copyright information and links in our footer. We use a grouped selector to target both our a:link and a:visited pseudo-classes, styling them black and switching off the links' default text-decoration (we'll indicate that they're links by including the URLs in a moment):

```
a:link, a:visited
{
color: black;
text-decoration: none;
}
```

The last thing we do, which we're introducing here for the first time, is to use CSS's :after pseudo-class, which will enable us to write a rule that displays the URLs of our links *after* the link text. This will be useful for anyone reading the King Kong page and wanting to find out a little more about the mighty ape online. We add the following:

```
a:link:after, a:visited:after
{
content: " (" attr(href) ") ";
}
```

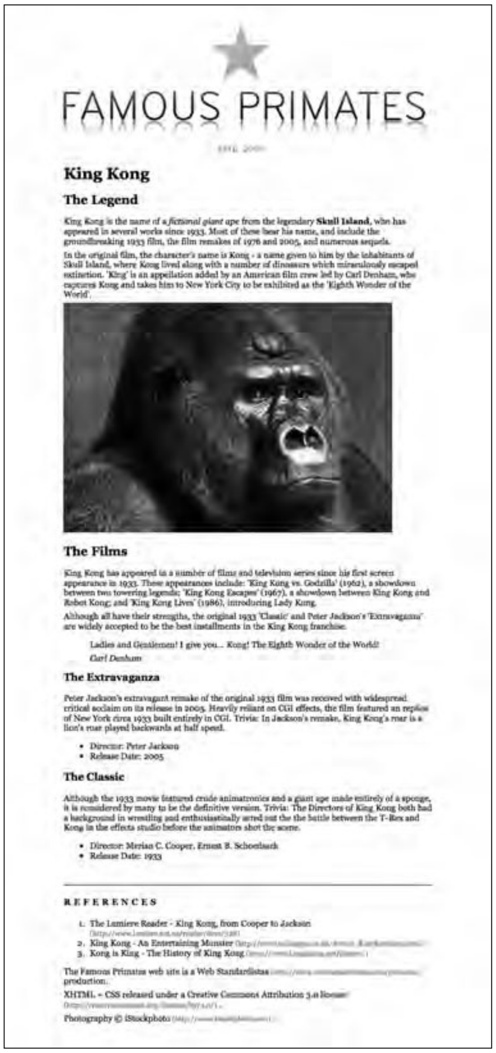
What this declaration does is to write some content to the page after our a:link and a:visited elements in standards compliant browsers. Essentially it writes the following: a space and an opening bracket, the href attribute, and a closing bracket and a space.

We add the following two declarations to our rule to style the URLs in a light shade of gray to reduce their prominence and set their `font-size` to 80%.

```
a:link:after, a:visited:after
{
content: " (" attr(href) ") ";
color: #CCCCCC;
font-size: 80%;
}
```

Click Print and check the results

That’s it, we’re finished. Simple. The only thing we need to do now is check the results when printed. Figure 13-5 shows the results of printing out our page. Perfect!



FAMOUS PRIMATES

EST. 2000

King Kong

The Legend

King Kong is the name of a *fictional giant* ape from the legendary **Skull Island**, who has appeared in several works since 1933. Most of these bear his name, and include the groundbreaking 1933 film, the film remakes of 1976 and 2005, and numerous sequels.

In the original film, the character's name is Kong - a name given to him by the inhabitants of Skull Island, where Kong lived along with a number of dinosaurs which miraculously escaped extinction. 'King' is an appellation added by an American film crew led by Carl Denham, who captures Kong and takes him to New York City to be exhibited as the 'Eighth Wonder of the World'.

The Films

King Kong has appeared in a number of films and television series since his first screen appearance in 1933. These appearances include: 'King Kong vs. Godzilla' (1962), a showdown between two towering legends; 'King Kong Escapes' (1967), a showdown between King Kong and Robot Kong; and 'King Kong Lives' (1986), introducing Lady Kong.

Although all have their strengths, the original 1933 'Classic' and Peter Jackson's 'Extravaganza' are widely accepted to be the best installments in the King Kong franchise.

Ladies and Gentlemen! I give you... Kong! The Eighth Wonder of the World!

Carl Denham

The Extravaganza

Peter Jackson's extravagant remake of the original 1933 film was received with widespread critical acclaim on its release in 2005. Heavily reliant on CGI effects, the film featured an replica of New York circa 1933 built entirely in CGI. Trivia: In Jackson's remake, King Kong's roar is a lion's roar played backwards at half speed.

- Director: Peter Jackson
- Release Date: 2005

The Classic

Although the 1933 movie featured crude animatronics and a giant ape made entirely of a sponge, it is considered by many to be the definitive version. Trivia: The Directors of King Kong both had a background in wrestling and enthusiastically acted out the the battle between the T-Rex and Kong in the effects studio before the animators shot the scene.

- Director: Merian C. Cooper, Ernest B. Schoedsack
- Release Date: 1933

REFERENCES

1. The Lumiere Reader - King Kong, from Cooper to Jackson [illegible]
2. King Kong - An Entertaining Monster [illegible]
3. Kong is King - The History of King Kong [illegible]

The Famous Primates web site is a Web Standardistas [illegible] production.

XHTML + CSS released under a Creative Commons Attribution 3.0 license [illegible]

Photography © iStockphoto [illegible]

Figure 13-5. Our King Kong page restyled for the wonderful world of paper

Conditional comments for Internet Explorer

Conditional comments only work in Internet Explorer and provide a mechanism for targeting specific versions of the browser. The conditional comment in our head example looks like this:

```
<!--[if lte IE6]>
<link rel="stylesheet" type="text/css" href="../css/iehacks.css" />
<![endif]-->
```

The conditional comment starts with a `<!--` and ends with a `-->`, so to all browsers other than IE the conditional comment will look just like one of the comments you met in Chapter 2 and be ignored. However, IE 6 and earlier versions of the browser will see the main style sheet we've linked to (`screen.css`) *and* the linked style sheet within the conditional comment (`iehacks.css`), which contains additional rules specifically written for IE that override the main style sheet.

It's worth noting that the reason the IE-specific rules in the `iehacks.css` style sheet override our `screen.css` style sheet is due to the fact that the `iehacks.css` link is below the `screen.css` link in our markup and so overrides any styles set in the `screen.css` style sheet. (To refresh your memory about this topic, you might like to revisit the section titled "The order of your CSS rules is important" in Chapter 10.)

As older versions of Internet Explorer notoriously suffered from shortcomings in the interpretation of CSS compared to browsers with better standards compliance, several methods of targeting CSS specifically at IE were developed. Known as **CSS hacks**, these workarounds took advantage of known bugs in different browsers' CSS interpretations to target rules to specific browsers.

One of the best-known CSS hacks is Tantek Çelik's box model hack, used to work around Internet Explorer 5's box model bug. To illustrate what a CSS hack looks like, we've included an example of this hack here:

```
#content
{
width: 400px;
voice-family: "\"}\"";
voice-family: inherit;
width: 300px;
}
```

The first declaration in the rule specifies a width for IE 5/Windows of 400px. What follows is designed to bamboozle IE 5 with a set of obscure rules that effectively force it to throw its hands in the air and give up on attempting to parse the remaining declarations. Smarter browsers, however, continue parsing and get the correct width of 300px on the last line of the rule. In a nutshell, IE 5 sets the width of the content to 400px, while smarter browsers set it to 300px.

If this looks complicated, it's because it is. As we said at the start of the example, this was a hack—a workaround web designers were forced into, as support for standards in older browsers evolved. While there might be times you're forced to use hacks, they should be avoided unless absolutely necessary.

Although a proprietary extension of regular HTML comments by Microsoft, conditional comments offer us a method of serving specific CSS to specific versions of Internet Explorer (arguably the number one culprit for browser bad behavior). Conditional comments allow us to write "good" CSS for all browsers and then add an additional set of amended rules for IE only.

This allows us to keep our main style sheet clean of obscure hacks and separate the IE-specific workarounds into a separate file. You can target specific versions of Internet Explorer using this method; in our example, in the typical head element we've used `<!--[if lte IE 6]>`, which targets versions *less than or equal to* IE 6 (`lte`).

The following example would target all versions of Internet Explorer 5 and above (conditional comments were introduced in IE 5, so they are supported from IE 5 onward):

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

If we just want to target IE 5 and IE 5.5, we could use the following, which targets versions of IE `lt` (less than) IE 6:

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

We can also use `gt` (greater than) and `gte` (greater than or equal to) as in the following example, which targets all versions of IE from 5.5 onward:

```
<!--[if gte IE 5.5000]>
<link rel="stylesheet" type="text/css" href="../css/iehacks.css" />
<![endif]-->
```

In the following section we'll look at a conditional comment in action as we use it to target a style sheet at IE 6 and earlier.

A conditional comment in action

In Chapter 10 we specified a 24-bit PNG image with a transparent background as part of our King Kong page. In the main style sheet (which we're now linking to) we've linked to this image. Unfortunately IE 6 doesn't natively support alpha transparency, resulting in unpredictable rendering. We can, however, use the conditional comment from our head element example to point IE 6 and below to `iehacks.css`, a style sheet built just for IE 6 and earlier where we address this problem:

```
<!--[if lte IE6]>
<link rel="stylesheet" type="text/css" href="../css/iehacks.css" />
<![endif]-->
```

We'll use this style sheet to set the transparent PNG used for all well-behaved browsers to a less pretty, but still workable, GIF image with a solid background color for IE. So, all other browsers will get `transparent_background.png` via the main style sheet:

```
/* Our main style sheet (screen.css) for well-behaved browsers. */

#header
{
background-image: url(../images/transparent_background.png);
}
```

The conditional comment will ensure IE 6 and older gets `solid_background.gif` via our IE hacks style sheet:

```
/* Our hacks style sheet (iehacks.css) for badly behaved browsers. */

#header
{
background-image: url(../images/solid_background.gif);
}
```

It's worth stressing that conditional comments, like hacks, should be used as a last resort; it's much better to find CSS solutions that work across all browsers. However, conditional comments are better than CSS hacks that rely on browser bugs; one reason for this is the ability to remove browser-specific CSS from your main style sheet, another is that any workaround rules are guaranteed not to have an impact on browsers other than those intended.

Adding a favicon

Adding a favicon (short for favorites icon) is by no means necessary; it is, however, the sort of attention to detail that the true Web Standardista should aspire to. Favicons appear in a number of places, including address bars, tabs, bookmarks, history, and browser toolbars. They provide an instant visual cue to the user, especially useful when scrolling back through browsing history, in addition to providing a valuable branding opportunity (admittedly small at just 16 × 16 pixels).

We add a favicon to our site using a `link` element as shown in the following example:

```
<link rel="shortcut icon" type="image/ico" href="../images/➥
favicon.png" />
```

Like the `link` elements we used for linking to our external style sheets, our link to our favicon contains a number of attributes and values: first, a `rel` attribute with a value of `shortcut icon`; second, a `type` attribute with a value of `image/ico`, informing the browser that the file being linked to is a favicon; lastly, an `href` attribute with a path to where the favicon is located in relation to the web page.

An example of a favicon is shown in Figure 13-6, at the Web Standardistas web site, where the URL is accompanied by a tiny gold star.

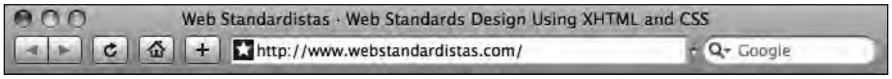

Web Standardistas · Web Standards Design Using XHTML and CSS
http://www.webstandardistas.com/
Google

Figure 13-6. Our Web Standardistas' favicon as it displays in Safari's address bar

Originally favicons had to be saved in an ICO format; however, a majority of browsers now support PNGs and GIFs too. You can even make animated favicons, although their support is limited at present.

Adding scripts

Although adding JavaScript to your page is beyond the scope of this book, we felt it important to introduce you to a `script` element so that you would know one when you encountered it on a typical web page when using View Source. Unsurprisingly, JavaScript is referenced using the `script` element as follows:

```
<script type="text/javascript" src="../js/primates.js"></script>
```

This example, from our typical head element introduced at the start of the chapter, shows a link to a JavaScript file called `primates.js` located in a folder called `js`.

It's worth noting that JavaScripts don't always need to be linked to; they can also be added between the opening and closing `<script>` tags as in the following example:

```
<!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>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>JavaScript Hello World!</title>
</head>
<body>
  <script type="text/javascript">
    document.write("Hello World!");
  </script>
</body>
</html>
```

This page, when loaded in a browser, will display identically to the very first "Hello World!" web page we created back in Chapter 2.

Contrary to what many beginning web designers mistakenly think, JavaScript is not Java and Java is not JavaScript. Both were initially developed in 1995, but both are different.

Java, developed by Sun Microsystems, is a full-fledged object-oriented programming language that can be used to create stand-alone applications in addition to mini applications, called applets.

JavaScript, developed by Netscape, is a smaller programming language commonly used to extend XHTML documents to provide levels of interactivity beyond what is possible with typically static XHTML pages. JavaScript is not used to create stand-alone applications or applets.

Testing and troubleshooting

We would be remiss if we didn't include a section on testing and troubleshooting. Both are important topics, and both will have an impact on your day-to-day progress as a Web Standardista.

By now you're well aware of the importance of testing your web pages using the W3C Markup Validation Service and CSS Validation Service, but by the very nature of the Web, and with the rapid emergence of the mobile Web, your web pages are likely to be seen in a variety of contexts, so testing in browsers is an equally important part of the testing process. The browser you're using isn't necessarily the browser your user is using, and as a consequence of this a thorough testing process should lie at the heart of your approach. A failure to test your web site can result in unexpected errors or worse, inaccessible content, a glaring Web Standardista faux pas.

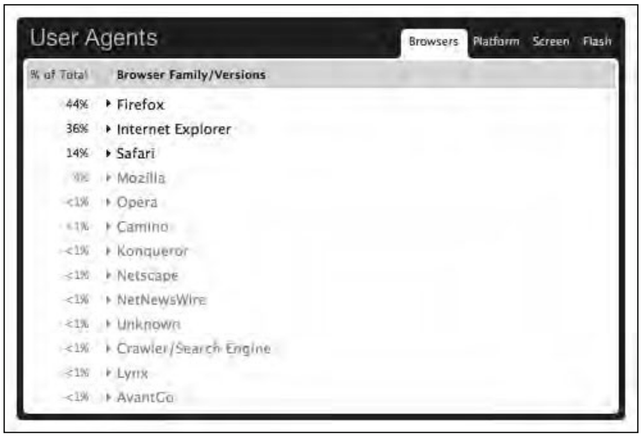
Thoroughly testing your web pages invariably highlights issues that will need troubleshooting if their cause isn't immediately apparent. To help you with this, we've highlighted a number of troubleshooting techniques in this section, a checklist if you will, that you can run through, prelaunch, to ensure any errors have been picked up and resolved.

Testing

The golden rule when developing and designing web sites is *test, test, test!* When you consider that your carefully crafted web site, overflowing with XHTML and CSS goodness, might be seen on anything from a laptop running Mac OS X using Safari, to a desktop running Windows Vista using Internet Explorer, to a tablet running Linux using a Mozilla-based browser, the potential for browser-related display issues becomes clear.

Mac OS X alone supports a variety of browsers including Safari, Firefox, Opera, and Camino (not to mention Lynx, which has reemerged as a Universal Access web browser for the visually impaired). One operating system, a variety of browsers. Windows and Linux are no different, supporting an extensive array of browsers from popular, well-established browsers to niche browsers.

Figure 13-7 shows a typical range of browsers as displayed in Shaun Inman's popular web site analytics program Mint (`www.haveamint.com`). Clearly the list is a long one with some of the usual suspects: Firefox, Internet Explorer, and Safari. However, it also includes a variety of lesser-known suspects: Camino, Konqueror, and Lynx, all clocking in at less than 1%.


User Agents

% of Total	Browser Family/Versions
44%	Firefox
36%	Internet Explorer
14%	Safari
0%	Mozilla
<1%	Opera
<1%	Camino
<1%	Konqueror
<1%	Netscape
<1%	NetNewsWire
<1%	Unknown
<1%	Crawler/Search Engine
<1%	Lynx
<1%	AvantGo

Figure 13-7. A typical list of user agents (or browsers) as shown in Mint

As you can see in Figure 13-7, not only do we need to factor in a variety of browsers when testing, we also need to consider screen size, not to mention the color depth of our users' screens. Testing for all of these eventualities is almost impossible, of course, so how do you test your web site across a variety of contexts? One answer lies in web-based browser test services, which allow you to see your web site through the eyes of another platform and browser.

Web-based browser test services

There are a number of web-based browser test services that, like the W3C validators, offer you a way of testing your web site. Essentially tools that allow you to see how your web site displays across a variety of operating systems and in a broad range of browsers, these range from paid-for services to open source initiatives that allow you to quickly test your web site in a variety of browsers. One that's well worth exploring is Browsershots. The description on the Browsershots web site describes it best:

Browsershots makes screenshots of your web design in different browsers. It is a free open-source online service created by Johann C. Rocholl. When you submit your web address, it will be added to the job queue. A number of distributed computers will open your website in their browser. Then they will make screenshots and upload them to the central server.

`www.browsershots.org`

Essentially, Browsershots uses an open source, distributed computing approach to create browser screenshots, splitting the workload across a number of distributed computers. Typically, the results take a few minutes to arrive upon submitting a URL; however, what you get over that few minutes is an extensive snapshot of how your web pages are looking in a variety of contexts.

Another service worth mentioning is Litmus (`www.litmusapp.com`)—a subscription-based browser and e-mail client test service. Through an elegant interface, shown in Figure 13-8, Litmus shows you exactly how your web site designs look on every platform, across every popular web browser. It also tests rich, HTML e-mail campaigns across a variety of different e-mail clients. Better still, once you've finished testing, a single click publishes a full compatibility report ready for review.

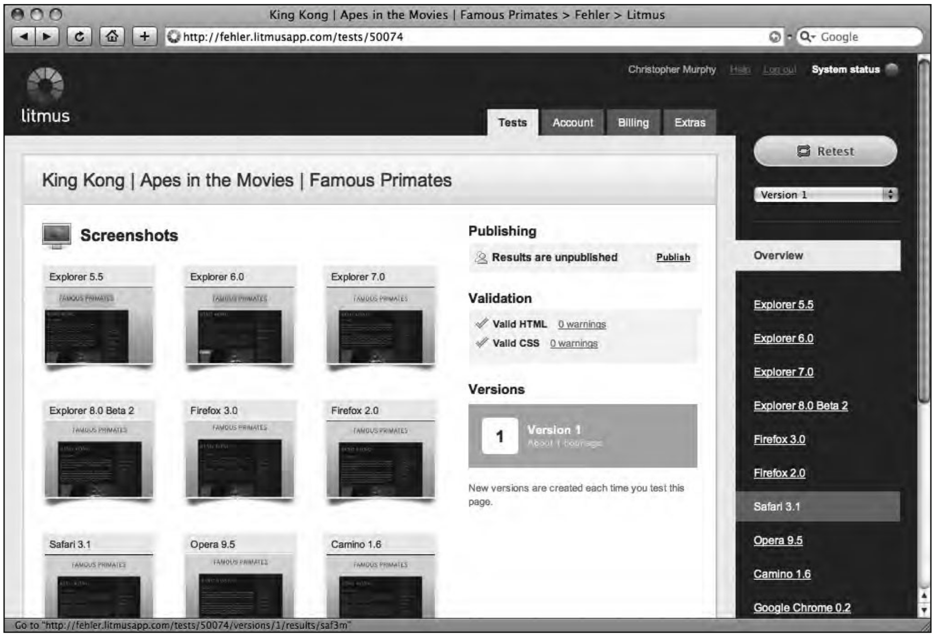


Figure 13-8. Litmus returns screenshots of the web page we've tested, allowing us to see how it looks across a variety of browsers.

Although these web-based browser test services are convenient and useful, they do, however, have one drawback. What they return is a static image, essentially only testing one state of your page and showing you the results. By their nature they can't show you how your web site loads and is displayed *over time*.

As you evolve as a Web Standardista and begin to embrace more advanced topics, for example, adding animation and scripting, or including dynamic, user-interface effects on your web pages, a service that returns a static image can't give you any insight into how these elements of your page are working.

There is a solution at hand, however: building a **guerilla testing suite**.

Building a guerilla testing suite

Given the focus of this book's homework on the Famous Primates web site, you might be forgiven for thinking a guerilla testing suite is one manned by a staff of apes. We can't be sure what you call your friends, but we do not call our friends apes (except under extreme circumstances). A guerilla (not gorilla) testing suite is a great way to build a low-cost, high-feedback testing suite for checking your web site.

Essentially an ad hoc testing service, manned by friends and fellow designers and developers using different platforms, a guerilla testing suite allows you to use your network of contacts to test out pages in a variety of situations.

Built on a trust-based model—if you test my pages, I'll test yours—this approach offers you a much deeper level of feedback: "Your dynamic, JavaScript-driven image rotator took forever to load!" or "There's a real problem with your images. There are far too many and they're far too large—I was able to boil the kettle and enjoy a pleasant cup of *Orange Pekoe* while I waited for the page to load."

This kind of critical and objective feedback is invaluable and is something that screenshot-driven services just don't offer. Even better, all it costs you is time.

Graded browser support

After reading the last few sections, you might be forgiven for wondering if you really need to support absolutely every possible combination of platform, operating system, and browser out there. The answer is that—in practical terms—you don't, especially if you embrace the concept of **Graded Browser Support**.

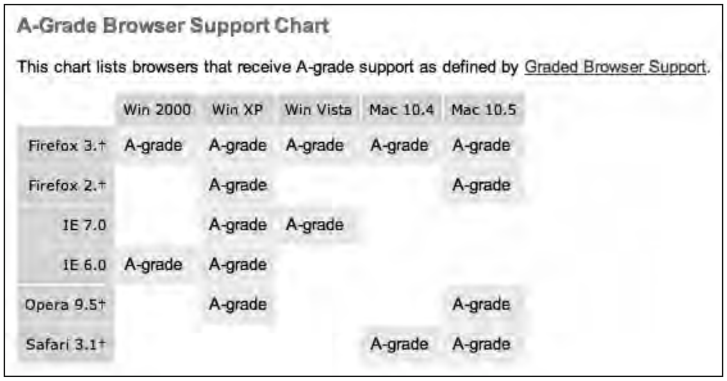
We have Yahoo! to thank for the concept of Graded Browser Support—essentially a broader and more encompassing definition of the word *support* coupled with the idea of grades of support. Yahoo! states the following about this concept:

> *In the first 10 years of professional web development, back in the early 90s, browser support was binary: Do you—or don't you—support a given browser? When the answer was "No", user access to the site was often actively prevented. . . . By contrast, in modern web development we must support all browsers. Choosing to exclude a segment of users is inappropriate, and, with a "Graded Browser Support" strategy, unnecessary.*
>
> `http://developer.yahoo.com/yui/articles/gbs/`

Essentially browsers are graded. Figure 13-9 shows Yahoo!'s current A-grade browser list. There are three grades: A-, C-, and X-grade, defined as follows:

- **A-grade**: A-grade support is the highest support level. By taking full advantage of the powerful capabilities of modern web standards, the A-grade experience provides advanced functionality and visual fidelity.

- **C-grade**: C-grade is the base level of support, providing core content and functionality. It is sometimes called **core support**. Delivered via nothing more than semantic XHTML, the content and experience is highly accessible, unenhanced by decoration or advanced functionality, and forward and backward compatible. Layers of style and behavior are omitted.
- **X-grade**: X-grade provides support for unknown, fringe, or rare browsers. Browsers receiving X-grade support are assumed to be capable. (If a browser is shown to be incapable—if it chokes on modern methodologies and its user would be better served without decoration or functionality—then it is considered a C-grade browser.)



A-Grade Browser Support Chart

This chart lists browsers that receive A-grade support as defined by Graded Browser Support.

	Win 2000	Win XP	Win Vista	Mac 10.4	Mac 10.5
Firefox 3.†	A-grade	A-grade	A-grade	A-grade	A-grade
Firefox 2.†		A-grade			A-grade
IE 7.0		A-grade	A-grade		
IE 6.0	A-grade	A-grade			
Opera 9.5†		A-grade			A-grade
Safari 3.1†				A-grade	A-grade

Figure 13-9. Yahoo!'s list of browsers that receive A-grade support

One important aspect of a Graded Browser Support strategy is to understand that not everyone gets the same experience. Expecting two users using different combinations of operating system and browser to have an identical experience fails to acknowledge the very essence of the Web, namely its diversity. Further, requiring the same experience for all users creates a barrier to participation—accessibility should be our key priority, as we've stressed throughout this book.

What does this mean in the context of troubleshooting and testing? Essentially, not everything will display as you expect it to all of the time, but that's to be expected. The Web is evolving, as you know by now, and things change. Quickly. Designing for the Web has always been a delicate balancing act between *progressive enhancement* and *graceful degradation*.

When designing for the Web, it's important to make an intelligent and informed decision about what you will support and what you won't. This is not a question of stating, "This web site is best viewed at 1024 × 768 or higher resolution with Microsoft Internet Explorer 6 or newer." We shudder when we read these declarations, and so should you. Tim Berners-Lee summarized this nicely in 1996 when he stated the following:

Anyone who slaps a "this page is best viewed with Browser X" label on a Web page appears to be yearning for the bad old days, before the Web, when you had very little chance of reading a document written on another computer, another word processor, or another network.

`www.anybrowser.org/campaign`

A far better approach is to point people in the direction of web standards, informing and educating them, an approach that lay at the heart of the WaSP's successful Browser Upgrade Campaign (`www.webstandards.org/action/previous-campaigns/buc/`), which began in 2001.

As a Web Standardista, you can do a great deal of good for the Web, helping to ensure that the road for future development is fundamentally standards based. At the end of the day, this book has been about enabling you to create sites that look fantastic in standards-compliant browsers. Spread that web standards message.

Troubleshooting

Yes, troubleshooting is frustrating, but isn't it helpful when a manufacturer has put a little thought into what might go wrong and suggested some possible techniques for turning those wrongs into rights? How many times have you returned home with your fresh-from-the-shop, shiny new breakthrough Internet communication device, only to discover it isn't quite working as you'd expected it to, right out of the box? You look for the instructions in frustration, hoping—praying even—that you'll find the word *Troubleshooting* on the contents page.

Good news, this is that section of the book (and we've even neatly listed it, clearly labeled "Troubleshooting," in the book's table of contents for future reference). This is the page you should be turning to when things are going wrong and you just can't even begin to work out why.

When something goes wrong, there's always a reason, of course, and finding it can be made a lot simpler by adopting a systematic approach using a few tried and tested techniques, which we introduce now.

Validate, validate, validate!

In Chapter 3 we introduced you to the W3C Markup Validation Service. When things go wrong, this should usually be your first port of call. Of course, if you've been a diligent Web Standardista, you should by now be using this validator as a matter of habit. Did you stray from the path? (Guilty? You know who you are!) If you did, learn to love this validator all over again. It is your friend.

Yes, the language used by the validator is a little dry and technical (and we'd love to meet the person who wrote "character "&" is the first character of a delimiter but occurred as data"), but, as we demonstrated in Chapter 3, the validator is the perfect tool for pinpointing any mistakes with laserlike precision:

Line 56, Column 8: character “&” is the first character of a delimiter but occurred as data.

Knowing that the possible source of the problem lies in Line 56 is a lot more helpful than knowing that the possible source of the problem lies in one of, say, 1,816 lines of markup.

It’s also worth noting at this point that the W3C Markup Validation Service isn’t the only free validation service the W3C offers. It also offers a CSS validator, which you’ve already seen, in addition to a number of others. If your web pages are passing the W3C Markup Validation Service, but still posing problems, test them by using the W3C CSS Validation Service. Like the W3C Markup Validation Service, the W3C CSS Validation Service helps you narrow down the potential source of problems, making the troubleshooting process easier. You can access the W3C CSS Validation Service here:

```
http://jigsaw.w3.org/css-validator/
```

Leanr to spel

Again, our next bit of advice sounds obvious and you should by now be used to this type of error, but spelling mistakes and accidental typos can often cause your carefully crafted web page to display in an unexpected manner. When writing markup by hand—a good thing as you now know—it’s possible for the occasional error to creep in here or there; we’re only human after all.

Checking your spelling can often highlight the cause of problems—again, the W3C’s validators are great tools for highlighting these issues.

The following is invalid:

```
a
{
text-decaration: underline;
}
```

The following is valid:

```
a
{
text-decoration: underline;
}
```

In this case a simple spelling mistake, `decaration` instead of `decoration`, has caused the problem. An easy mistake to make, but one that is often overlooked. Looking on the bright side, writing your markup and CSS by hand in a plain text editor will improve your spelling and attention to detail, and that’s no bad thing.

Another spelling-related issue that can occur from time to time—again an easy mistake to make—is when a word is spelled correctly in HTML, for example, `<div id="content">` but spelled incorrectly in the corresponding CSS, for example, `#contnet {color: red;}` or vice versa. Again, the result is an error; frustrating, but understandable when you find it. Case sensitivity can also be an issue, for example, `#Content` and `#content` are not the same.

Finally, it's worth mentioning that mistakes can also crop up even when you've diligently checked all of your spelling. The following example, where the HTML id doesn't match up with the CSS id, might cause a few sleepless nights. This HTML:

```
<div id="nav">
```

will obviously not be styled by this CSS:

```
#navigation
{
...
}
```

The bottom line when mistakes occur is to check everything carefully, using the validators to help you find the needles in the haystack.

Adopt a lurid palette

Another useful technique, especially when troubleshooting CSS layout issues, is to temporarily adopt a lurid palette of background colors to clearly highlight the different sections or divs within your document. Using a distinctive color scheme, setting background colors to bright, easily distinguishable colors—for example aqua, fuschia, lime, red, and yellow—allows you to clearly indicate the areas your different divs are occupying and their relationship to each other.

This can also be a useful approach for other elements as well, for example, applying a background-color to text elements to highlight them and clearly indicate how much space they occupy on the page. (We used this technique in Chapter 10 when we introduced margins, borders and padding on a simple p element.)

Check for repetition

In Chapter 10 we explained how rules lower in a style sheet override rules targeting the same element that occur above them. This might not seem to be a likely occurrence in a short page, but as your style sheets get longer and more complex, it's an easy mistake to make. The h1s in the following example will appear red, as the rule styling h1s in red appears lower than the first rule, which styles the h1s in blue.

```
h1
{
color: blue;
}

/* Imagine lots of additional rules here. */

h1
{
color: red;
}
```

Using your plain text editor's Find command can help pick these sorts of mistake up quickly.

Reduce to deduce

When all else fails: *reduce to deduce*. Simplify your XHTML and CSS by selectively removing parts of your code from the picture, and then refresh your web page in the browser. Removing CSS rules one by one will usually help identify the causes of any problems. Before embarking on this process, however, it's a good idea to make a backup copy of the file you're testing.

An easy way to achieve this is through the use of comments to selectively switch off aspects of your XHTML or CSS (a process known as **commenting out**). The following example shows a rule styling our h2s that has had the line-height commented out:

```
h2
{
font-size: 24px;
/* line-height: 0.6em; */
margin: 0;
padding: 26px 0 10px 0;
}
```

Commenting out the line-height in this example and refreshing the page in the browser allows us to quickly test the effect that this declaration has on our markup.

XHTML rule reference

To assist you with your troubleshooting, we've included a ready reference of XHTML rules in this section.

Remember that adopting XHTML Strict forces us to use stricter rules that are easy to forget. Your mind wanders, a mistake creeps in, and as a consequence your page throws up a glaring error. Although we've covered these rules throughout the book, we've listed them again here for easy reference:

- Ensure you've opened with the proper DOCTYPE and namespace (if you've been working from the file we provided at the book's companion web site—www.webstandardistas.com/02/template.html—you should be fine).
- All markup must be written in lowercase: `<p>...</p>` is valid, `<P>...</P>` isn't.
- Every tag you open must close. "Empty" elements—`<img ... />`, `<br />`, `<meta ... />`, and `<link ... />`, for example—must be closed with a space and slash—" `/>`".
- Nesting must be symmetrical. Remember the First In, Last Out rule: `<p><strong>pioneer</strong></p>` is valid, `<p><strong>pioneer</p></strong>` isn't.
- Encode all <, >, and & characters. Less than or more than signs—(<) and (>)—that aren't part of a tag must be encoded as `<` and `>`, respectively; likewise ampersands (&) must be encoded as `&`.

- All attribute values must be enclosed in quotes: `width="500"` is valid, `width=500` isn't. You also need to separate attributes with spaces: `width="500" height="375"` is valid, `width="500"height="375"` isn't.
- Don't put double dashes within comments: `<!-- Bananas -->` is valid, `<!-- Fruit -- Bananas -->` isn't.

Summary

So what have we covered? In this chapter we've tied up a lot of loose ends. The most significant thing we've covered is how to move from an embedded style sheet, useful during the design, development, and build phases, to an external sheet that maximizes the real benefits of a CSS-based approach: strength, power, and flexibility.

We've also looked at the contents of a typical head element, introducing you to `<meta>` tags, exploring how they can be used to provide additional information about our web pages; the `link` element, looking at how it can be used to link to external style sheets and to include a favicon for our web pages; and lastly the `script` element, acquainting you with it and briefly showing an example of JavaScript in action to create a simple JavaScript "Hello World!" page.

Finally, we took a look at the importance of testing and troubleshooting, giving you a troubleshooting checklist that you can refer to if things go wrong.

In the next chapter we ask, "Where to from here?" and answer that question with some pointers—a few suggestions for what you might focus on after you've completed this book.

Homework: Linking to external style sheets

In this chapter we introduced you to the typical contents of a head element to highlight some additional head elements you might encounter as you use View Source to view web pages "in the wild."

We used our journey through the head to introduce you to a number of elements you might find useful, not least the `<meta>` tags we introduced. We also used this journey to demonstrate the power of putting external style sheets to work, introducing you to style sheets for both screen and print media.

As we had reached the culmination of the design process, we took our completed internal style sheet and showed you how to use it as the basis for creating an external style sheet. This will allow you to take full advantage of the benefits of using a *single* style sheet that you can link *all* of your web pages to. We also introduced print style sheets that allow you to create perfectly printed web pages optimized for the medium of print.

Along the way we introduced you to favicons, explaining how they can prove useful as a usability aid. Finally, we introduced you to a number of techniques and tools that can prove useful in troubleshooting your web pages.

Your homework for this chapter will be to apply what you've learned to your Gordo page, adding to its `head` element and creating an external style sheet that you can then use to style all of your Famous Primates pages.

1. Add some <meta> tags

Using the `meta description` tag we created for our King Kong web page as a guide, create a `meta description` tag for your Gordo page. We've left it to your discretion to write the contents of the `description` attribute; however, bear in mind that the first words you use are the important ones. Search engines will often truncate longer `meta descriptions`, so make your first words count.

You can find out a little more about writing good-quality `meta description` tags here:

```
http://tinyurl.com/mrgoogle
```

Once you've added your `meta description` tag, add a `meta name` tag and insert your name in the `content` attribute, as in the following example:

```
<meta name="author" content="Your Name" />
```

2. Create an external style sheet

In your plain text editor, create a new document and save it as `screen.css`; you'll transfer the rules you've written up to this point on your Gordo page to this document. Referring to the examples covered in this chapter, remove the rules you've written for your Gordo page and add them to your `screen.css` file.

Create a `css` folder for your screen and print style sheets; this is where you'll store your CSS files. Add a `link` element to the `head` element of your page and create a link to your brand-new CSS file.

3. Link to a print style sheet

We've provided a ready-made print style sheet for you to link to. You can download the print style sheet (along with a number of other assets) from the following location:

```
www.webstandardistas.com/13/assets.zip
```

Once you've downloaded these files, transfer the `print.css` file into your `css` folder and add a link to it. Remember to include a `media` attribute for both of your linked style sheets to inform the browser of their purpose.

4. Add a favicon

You'll see that we've provided a favicon for you, which was included with the other assets for this chapter. Referring to our example, add a link to your new favicon and test it in your browser.

5. Test and troubleshoot

Using your Gordo page as an example page, test out the page using both Browsershots (`www.browsershots.org`) and Litmus (`www.litmusapp.com`). This will give you some idea of how others see it. Needless to say, we expect you to take care of any troubleshooting that might be required (you can use our checklist to assist you in this process).

As before, to help you with creating your external style sheet, we've created our own, similarly styled, page about King Kong featuring all of the additional material we covered in this chapter. You can refer to this, using your browser's View Source menu command to see how we've created our external style sheet, here:

```
www.webstandardistas.com/13/king_kong.html
```

Once you've created your external style sheet, linked to the print style sheet we supplied, and tested it out by printing your Gordo page, put the kettle on and enjoy a cup of *Assam SFTGFOP Mangalam* as you prepare yourself for the next chapter.