Picking a Rendering Mode

Standing in the middle of yesterday
Where it all went wrong—where we made mistakes
I'm sorry for the things I forgot to say
But it won't be long until it will be okay

—*Raine Maida*

Suppose you've spent a few years and several million dollars developing a product that rapidly scans Dewey Decimal numbers on book spines and sends those numbers to a central database. This enables libraries to keep track of what they physically have on hand. You sell your product to hundreds of libraries all over the country and get a lot of rave reviews.

Then one day a large number of libraries decide to abandon Dewey and go to an alternate system, one that allows for more expansion. Many of your customers will be making this switch, but they still want to use your device. They're willing to pay for an upgrade, and you could provide one, but if you change the product to use this new system, it won't read the Dewey numbers anymore. That would prevent your other, Dewey-based clients from buying the upgrade and would turn away some new customers.

The simple answer is to build both systems into the device and put a switch on the side so that users can pick which scanning mode they want. This gives you a more flexible device that doesn't turn away any customers.

So, what does this have to do with CSS? To answer this question, let's take a quick look backward.

A Short History of Incompatibility

One of the great tragedies in the development of the Web was the browser wars that most digital historians agree began in earnest in 1997. There is some disagreement over whether the conflict ever ended, although many feel that active combat ceased somewhere around late 1999 or early 2000. The greatest casualty of this conflict was the interoperability that was the Web's foundation. In the quest to create the "killer app," browser companies devised one proprietary feature after another.

At the height of the conflict (primarily 1998 and 1999), the two main combatants were Netscape Navigator 4 and Internet Explorer 4. These two browsers had completely different document object models, differences in their handling of HTML layout and whitespace, and a vast slew of bugs, shortcomings, and flaws in their support for CSS. In desperation, many developers turned to serving a different style sheet for each browser rather than trying to navigate the minefield of incompatibilities that had been created.

If one or the other of these browsers had been standards compliant, designers might have had a fighting chance, but sadly that was not so. The effort to accommodate browser bugs taught legions of designers bad habits and promoted thinking that ran counter to the W3C specifications. Even worse, it made their documents a soup of tricks that would never validate and that were intended (consciously or otherwise) to only work in the browsers that existed at the time.

All of this left the battlefield littered with the sanity of more than one designer. When Internet Explorer 5.0 was released on Windows, things didn't get any better. Although it was moving forward and expanding the model laid down by IE4, Netscape was regrouping and not releasing anything new. The drastic limitations of Navigator 4 were still of primary concern and prevented the adoption of many a cool new trick.

Enter the Future of the Web

It was the release of Internet Explorer 5.0 for Macintosh that first pointed to a way out of the morass that the browser wars had created. Programmers for IE5/Mac recognized that no browser could afford to break old pages. To permit a move to standards-based markup, the very behaviors on which the old pages

were based would have to be broken. The solution was to implement both a standards-compliant rendering engine and the old, "bugwards-compatible" behaviors…and then provide a mechanism that would let the author of the document choose which rendering mode the browser should use in displaying the document.

Several mechanisms were considered, but the one that seemed to make the most practical sense was the DOCTYPE that all documents are *supposed* to contain. In theory, every HTML document should declare its document type using a directive at the very top of the file. For example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
        "http://www.w3.org/TR/REC-html40/strict.dtd">
<html>
```

In this example, the document has been marked as using strict HTML 4.0. There are many DOCTYPEs, some of which are listed in Table 1.

Table 1 A Sampling of **DOCTYPE** Values

Document Type	DOCTYPE
HTML 3.2	`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">`
HTML 4.0 Transitional	`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">`
HTML 4.0 Frameset	`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN" "http://www.w3.org/TR/REC-html40/frameset.dtd">`
HTML 4.0 Strict	`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">`
HTML 4.01 Transitional	`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">`
HTML 4.01 Strict	`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">`
XHTML 1.0 Strict	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/xhtml1-strict.dtd">`

What DOCTYPE?

As of this writing, it is the case that the majority of Web documents contain no DOCTYPE at all.

URL or URI?

Although most Web authors are familiar with the acronym URL, which stands for uniform resource locator, the term URI isn't as common. URI stands for uniform resource identifier. The primary difference is that whereas a URL must point to a resource on the Web, a URI does not have this restriction. However, URIs must be unique. An analogy would be the word "heaven." It may or may not have a physical location, but either way it describes a unique concept.

As you can see in Table 1, some of the DOCTYPEs have URIs and some do not. This is not a hard rule: Any DOCTYPE can have a URI or leave it off. Thus, they were included at random in the examples shown in the table. As you'll see later, however, the presence or absence of a DOCTYPE URI can affect which rendering mode gets picked.

The mechanism of DOCTYPE *switching* is, at its core, fairly sensible and straight forward:

- Documents with older or Transitional DOCTYPEs (or no DOCTYPE at all) are displayed using the "loose rendering" or "quirks" mode. This mode, also called "bugwards" compatibility, emulates legacy bugs and behaviors of version 4 browsers.
- Documents with Strict or XHTML DOCTYPEs are displayed using the "strict rendering" mode. This mode follows W3C specifications for HTML, CSS, and other layout languages as closely as possible.

Although incredibly useful for authors, DOCTYPE switching might have remained no more than a curiosity had it only been implemented in IE5 for the Macintosh. Happily, Netscape 6 and Internet Explorer 6 for Windows have since adopted it. The door to a standards-compliant Web stands wide open.

DOCTYPE Switching in Detail

And the Other Browsers?

Netscape Navigator 4.x came long before DOCTYPE switching was even conceived, so it should be assumed that it's always in quirks mode (and a buggy form of it at that). Opera 6 and earlier versions do not bother with DOCTYPE switching and should be assumed to be in strict mode. Note that Opera might still have bugs, but its behavior is very close to the strict modes of other browsers.

If you're going to step through a door, it's usually a good idea to know something about what's on the other side. It's also a good idea to know how to get through the door in the first place. To that end, Table 2 provides a sampling of DOCTYPEs and the effect they'll have in the various Web browsers that recognize DOCTYPE switching at all.

Table 2 DOCTYPE Switching in Various Browsers

DOCTYPE	IE6/ Win	IE5/ Mac	NS6.0/ NS6.1/ NS6.2
No DOCTYPE provided	Q	Q	Q
Unknown DOCTYPE	S	S	Q
HTML 2.0	Q	Q	Q
HTML 3.2	Q	Q	Q
HTML 4.0 Frameset	Q	Q	Q
HTML 4.0 Frameset + URI	S	S	Q
HTML 4.0 Transitional	Q	Q	Q
HTML 4.0 Transitional + URI	S	S	Q
HTML 4.0 Strict	S	S	S
HTML 4.0 Strict + URI	S	S	S

DOCTYPE	IE6/ Win	IE5/ Mac	NS6.0/ NS6.1/ NS6.2
HTML 4.01 Frameset	Q	Q	Q
HTML 4.01 Frameset + URI	S	S	S
HTML 4.01 Transitional	Q	Q	Q
HTML 4.01 Transitional + URI	S	S	S
HTML 4.01 Strict	S	Q	S
HTML 4.01 Strict + URI	S	S	S
Any known XHTML	S	S	S
Any known XHTML + URI	S	S	S

S = Strict mode Q = Quirks mode

Differences in Rendering Modes

If you plan to upgrade your old pages to new markup and a new `DOCTYPE`, it will help to know what changes you're likely to encounter. There are profound changes such as an altered meaning for the properties `width` and `height`, and there are subtle changes like inheritance into tables that can still wreak havoc with legacy designs. There might even be differences in the way the CSS can be written, depending on the browser.

The following information, while not a comprehensive list of every last difference between quirks and strict modes in various browsers, is an attempt to touch on the areas most likely to cause an author trouble.

Inheritance and Tables

The biggest area of potential trouble relates to tables and their inheritance (or lack thereof) of styles. In older browsers such as Navigator 4.x and Internet Explorer 5.x (and earlier), styles such as fonts and font sizes were not inherited into tables. Consider the following simple test case:

```
<body style="font: large sans-serif; color: purple;">
<table>
<tr><td>Hey, it's text in a table!</td></tr>
</table>
```

In old browsers, the text within the table would have been neither large nor in a sans-serif font. In old versions of Explorer, the text would not even be purple; instead, it would remain the user's default text color (usually black).

In quirks mode, this lack of inheritance is preserved. In strict mode, all styles are inherited by text within tables. This can actually lead to trouble due to another legacy bug, this one in Navigator 4.x.

Let's say an author wanted the text on his page to be smaller than the user's default. In most browsers, you can get that effect by writing something like this:

```
body {font-size: 0.8em;}
```

However, because the value of `font-size` (among others) was not inherited in tables, a common workaround was to assign the same values to table cells.

```
body, td, th {font-size: 0.8em;}
```

This made font sizes basically consistent throughout the document in 1998-era browsers. Unfortunately, in modern browsers, the preceding rule will make text inside of tables a maximum of 64% of the user's default and possibly smaller!

This happens because when a browser allows properties like `font-size` to inherit (as it should), you have a situation in which a table cell, which is a descendant of the body element, has its text set to `0.8em` of `0.8em` of the user's default `font-size` setting, which yields `0.64em`. If there is a table nested within a table, its text will be `0.64em` times `0.8em`, or `0.512em`. That's a shade over half the user's default font size!

An easy way to see this sort of effect is to set up a number of lists nested inside each other, going to at least three levels of nesting.

```
<ul>
<li>list item
<ul>
      <li>sublist item
            <ul>
            <li>subsublist item</li>
            </ul>
      </li>
      </ul>
</li>
</ul>
```

Now add to this document the style `ul {font-size: 0.8em;}`. Each level of nested list will get smaller and smaller, just as nested tables will do when inheritance works properly.

Reducing the Font

The practice of making a page's text smaller than the user's default is generally thought to be a poor authoring practice. Unfortunately, browser defaults are usually larger than most designers would like. There is no perfect answer, but think carefully about reducing font sizes because you might make the text too small for the user to read comfortably.

▸▸ The Sad Story of Tables

Upon making the switch from legacy authoring and the styling of tables, many an author is inclined to ask why tables were so badly broken in older browsers. There were many reasons, not the least of which was that although the browsers were good efforts for their time, they've come to be seen for what they were: rush jobs to cram many new "features" into the browsers so that they looked much better than the previous versions. As a result, these browsers weren't well engineered, and designers have paid the price for all the years since then.

In the case of Navigator 4.x, the culprit was the rendering engine itself. Utilizing the same codebase that had been evolving ever since Netscape 1.0, the rendering engine in NN4.x was starting to buckle under its own weight. In a certain sense, NN4.x treated tables almost as if they were separate documents that had been inserted into the main document—and this caused inheritance to fail.

In Explorer's case, the engine was new but the thinking behind it wasn't. What happened here was that the browser attempted to enforce certain default styles on tables. It was almost as if the browser kept internal styles that read something like this:

```
table {font-size: [[default_user_font_size_setting]];
       color: [[default_user_text_color_setting]];}
```

So, in a typical browser installation, table text would always be black and 16 points in size. (It would inherit the `font-family`*, though, strangely enough.)*

Whatever the rationale, the fact remained that styling tables became a major source of authorial pain, and the loud complaints over this issue helped encourage browser vendors to push much harder toward standards compliance.

Case Sensitivity

In the HTML 4.01 specification, `class` and `id` values are defined to be case sensitive; that is, they must have the same capitalization. In other words, `Hello` and `HelLO` are not the same thing. Thus, you can run into situations such as the following:

```
<html><head><title>Case Sensitivity</title>
<style type="text/css">
p.TestThisClass {color: red;}
</style>
</head>
<body>
<p class="testthisclass">This text isn't red!</p>
</body>
</html>
```

The Special Case of `id`

Although `id` values are case sensitive, there can be no case insensitive matches within an HTML document. In other words, although `TestID` and `testid` are not the same thing, only one of them can be found within the source of a given document. The same is true of `name` values, as it happens.

Browsers from the version 4 era treated `class` and `id` values as being case insensitive, which would make the text in the preceding code block red. Modern browsers in strict mode will not color the paragraph red.

Because there is no penalty for making sure that all of your CSS rules and HTML-based values for `class` and `id` have the same case, you should always make sure the case matches between the two.

`class` and `id`, Take Two

There were some other oddities concerning `class` and `id`. Somewhat strangely, it was not permissible under CSS1 to begin `class` and `id` values with a digit (0–9), but some browsers allowed you to do this. Modern strict-mode browsers will very likely ignore any `class` or `id` value that begins with a digit, so if you have an `id` called `1st`, you'll need to rename it.

Avoid Underscores

Because of their twisted support history, you are strongly encouraged not to use underscores in `class` and `id` values. Hyphens are permitted and are a common substitute (as in test-class).

Another weirdness deserves mention even though it isn't precisely a strict/quirks issue. Both CSS1 and CSS2 did not permit the use of underscores in `class` and `id` values, which made values like `test_class` illegal. Explorer has always allowed underscores, but Navigator 4.x did not. After CSS2 was published, some errata were added, and one of those was to allow underscores. Although this means that most browsers are now compliant and allow underscores, some do not.

Value Problems

This section belongs entirely to Internet Explorer. In IE4.x/Win and IE5.x/Win, you were able to write fairly sloppy values. For example:

```
h1 {color: FF0000;}     /* missing the octothorpe! */
h2 {font-size: 18 px;}  /* one too many spaces */
table {width: 500;}     /* where's the unit? */
```

A Quick Aid To Learning

One of the fastest ways to catch errors in your CSS, and thus learn good authoring habits, is to use a CSS validator. These are programs that check your CSS for syntax errors and common mistakes. The W3C offers a good one at `http://jigsaw.w3.org/css-validator/`. Note that validators will not prevent you from making all possible mistakes, but they will help you avoid basic errors.

In the first rule, the color value is missing the octothorpe (`#`) that is required in front of any hexadecimal color value. The second rule has an invalid space between the number `18` and the unit `px`, and this entirely changes the meaning of the value. For the third rule, there is no unit at all. Should the `table` be 500 furlongs wide? 500 stadia? 500 angstroms?

The correct forms of these rules is as follows:

```
h1 {color: #FF0000;}
h2 {font-size: 18px;}
table {width: 500px;}
```

Internet Explorer 6 in strict mode will properly ignore the first set of rules as incorrect. If you've spent the last year or three learning bad authoring habits because they worked in Explorer, it's time to unlearn those habits.

Changing Width and Height

This is by far the change that will be the biggest surprise to authors who are making the switch from legacy to standards-compliant layout…especially those authors who did any CSS positioning or attempted pixel-precise layout of elements in Explorer 5.x.

This topic is best described by using diagrams. In Figure 1, you see the element box as described in CSS1 and CSS2.

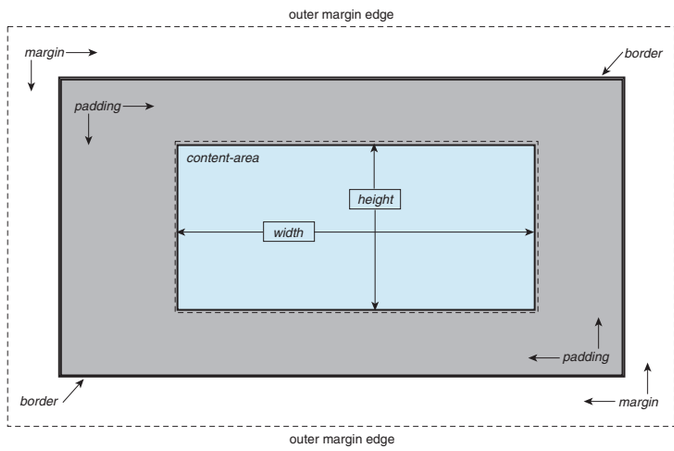


FIGURE 1

The CSS element box.

Notice that the properties `width` and `height` describe the size of the content-area. If there is any padding or borders, these are added to the content-area. For example, assume the following:

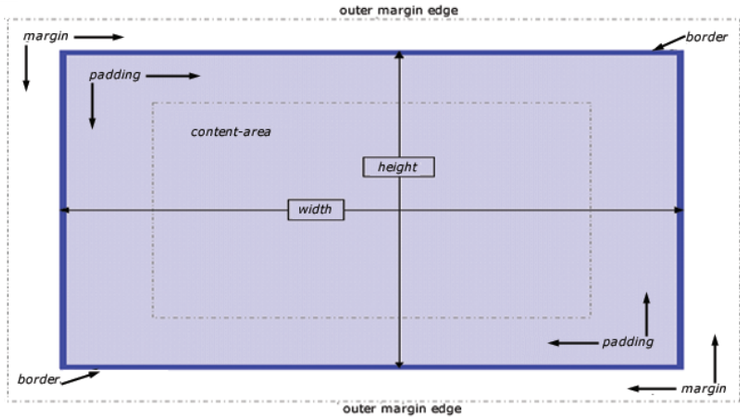
```
div.illus {width: 200px; height: 150px;
   padding: 20px; margin: 10px; border: 3px double black;}
```

As per the preceding styles, the content-area is 200 pixels wide by 100 pixels tall. The distance from the outer edge of the left border to the outer edge of the right border is 246 pixels (3 + 20 + 200 + 20 + 3). From the left outer margin edge to the right is 266 pixels, after you add 10 pixels of margin on each side. Similarly, the distance from the top border edge to the bottom is 196 pixels and from the top margin edge to the bottom is 216 pixels.

Now examine Figure 2, which shows the Internet Explorer 4.x and 5.x element box.

FIGURE 2

The Internet Explorer (legacy) element box.

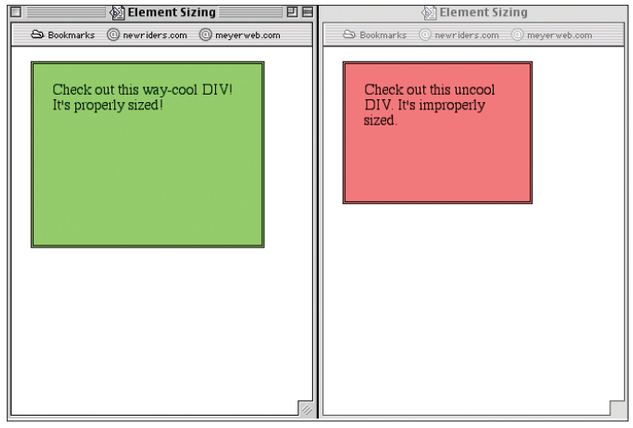


Note the differences in the definition of `height` and `width`. In old Explorer versions, these property values described the aggregate of the content-area, padding, and borders instead of just the content-area.

Therefore, given our previous styles, the content-area would be 154 pixels wide and 104 pixels tall. The distance between the outer border edges would be 200 pixels horizontally and 150 pixels vertically, and the distance between outer margin edges would be 220 pixels by 170 pixels. The differences are shown side by side in Figure 3.

FIGURE 3

The two element sizing models compared: standards-compliant on the left, old Explorer model on the right.



Regardless of which approach makes more sense to you, the correct element box model is the CSS model, and that's the one used by IE6 when in strict mode. In quirks mode, it uses the old Explorer model. This means that, simply by changing the `DOCTYPE` of a page, you can make IE6 radically alter a page's layout without changing a single character of your CSS.

Because these two models are so different, this might seem to be an intractable problem for any page that needs to be viewed in multiple browsers. This is not necessarily the case. There are ways to avoid layout trouble by carefully constructing your document structure (see Project 11, "Positioning a Better Design") and by using the flaws in old-style parsers to serve up browser-specific CSS without using JavaScript to do it (see "Tricking Browsers and Hiding Styles" on the Web site).

Picking Your Favorite Model

The subject of which layout model makes more "sense" has raged for years now and shows no sign of ever being settled. Nonetheless, there is hope for both camps: There is a property proposed for inclusion in CSS3 that will enable the author to decide which model to use when sizing an element. As of this writing, the property was called `box-sizing`, but be aware that name could change before the CSS3 box model is finalized.

Further Reading

If you are interested in reading more about `DOCTYPE` switching and the differences between rendering modes or if you want to peruse more detailed tables of `DOCTYPE`s and learn what modes they trigger in which browsers, here are some good resources to try out.

CSS Enhancements in Internet Explorer 6

`http://msdn.microsoft.com/library/en-us/dnie60/html/cssenhancements.asp`

This document provides an introduction to the concept of `DOCTYPE` switching, a table summarizing how to trigger quirks or strict mode in IE6, and relatively detailed explanations of the differences between the two modes.

Mozilla's Quirks Mode

`http://www.mozilla.org/docs/web-developer/quirks/`

This small collection of documents explains the rationale behind `DOCTYPE` switching in Mozilla, provides a thorough listing of which `DOCTYPE`s will get you into which mode, and offers a list of quirks. In many cases, the quirks link to entries in Bugzilla, the Mozilla bug-tracking system, and provide an occasionally fascinating insight into debates about these quirks and whether or not they needed to be fixed at all.