# Activating Browser Modes with Doctype

In order to deal both with content written according to Web standards and with content written according to legacy practices that were prevalent in the late 1990s, today’s Web browsers implement various engine modes. This document explains what those modes are and how they are triggered.

## Summary for the Impatient

The main conclusion to draw from this article is that you should start all your HTML documents (i.e. anything that gets served as text/html) with <!DOCTYPE html> as the first thing in the source. (Read on to learn why.)

If you want to take extra care to make sure that users of IE8, IE9 or IE10 cannot press a button that makes your site regress as if it was being viewed in IE7, either configure your server to send the HTTP header X-UA-Compatible: IE=Edge for text/html or put <meta http-equiv="X-UA-Compatible" content="IE=Edge"> in the head of your HTML documents (before any scripts). This will, however, make the HTML document invalid and if you don’t include these IE-specific incantations, the default behavior of IE is reasonable in most cases, so you don’t really *need* to jump through these IE-specific hoops. (Read on for exceptions.)

## The Scope of This Document

This document covers mode switching as it applies to Firefox and other Gecko-based browsers, Safari, Chrome and other WebKit-based browsers, Opera, Konqueror, Internet Explorer for Mac, Internet Explorer for Windows (including Windows Phone) and browsers that embed IE. Instead of referring to the [names of browser engines](http://hsivonen.iki.fi/engines/), the names of the best-known browsers using each engine are used instead.

This document focuses on the mode selection mechanism and does not document the exact behaviors of each mode. The purpose of this document is to give you information on how to avoid the legacy modes. The purpose of this document is not to facilitate cherry-picking behaviors by deliberately choosing legacy modes.

## The Modes

Here are the various modes:

### Common Modes for text/html Content

The choice of the mode for text/html content depends on doctype sniffing ([discussed later in this document](https://hsivonen.fi/doctype/#sniffing)). In IE8 and later, the mode also depends on other factors. However, by default even in IE8 and later, the mode depends on the doctype for non-intranet sites that are not on a blacklist supplied by Microsoft. Additionally, there are other factors involved even with IE6 and IE7 if Google Chrome Frame has been installed.

Quirks Mode

In the Quirks mode the browsers violate contemporary Web format specifications in order to avoid “breaking” pages authored according to practices that were prevalent in the late 1990s. Previously, different browsers implemented different quirks. In particular, in Internet Explorer 6, 7, 8 and 9, the Quirks mode is effectively frozen IE 5.5, while in other browsers the Quirks mode has been a handful of deviations from the Almost Standards mode. Recently, browsers have been converging on common behavior in their Quirks modes. Most notably, the primary Quirks mode of IE10 is no longer an imitation of IE 5.5 but seeks to be interoperable with Quirks modes of other browsers. The[interoperable quirks are being standardized](http://quirks.spec.whatwg.org/) at the WHATWG.

The primary Quirks mode of IE10 that is like the Quirks modes of other browsers is sometimes referred to as the “interoperable Quirks mode” to distinguish it from the IE 5.5-imitating “Internet Explorer 5 Quirks” mode that IE10 also has.

If you are authoring new page, don’t use the Quirks mode. Just don’t. Use the Standards mode.

Standards Mode

In the Standards mode the browsers try to give documents the specification-wise correct treatment to the extent implemented in a particular browser.

Since different browsers are at different stages of compliance, the Standards mode isn’t a single target, either.

The [HTML specification](http://www.whatwg.org/specs/web-apps/current-work/multipage/) calls this mode the “no quirks mode”.

Almost Standards Mode

Firefox, Safari, Chrome, Opera (since 7.5), IE8, IE9 and IE10 also have a mode known as “the Almost Standards mode”, which implements the vertical sizing of table cells traditionally and not according to the CSS2 specification. Mac IE 5, Windows IE 6 and 7, Opera prior to 7.5 and Konqueror did not need an Almost Standards mode, because they didn’t implement the vertical sizing of table cells according to the CSS2 specification in their respective Standards modes anyway. In fact, their Standards modes are closer to the Almost Standards mode than to the Standards mode of newer browsers.

[In retrospect](http://hsivonen.iki.fi/almost-precedent/), the Web would have been better off by not having the distintion between Standards and Almost Standards, having the Almost Standards behavior as the default and having a CSS property for opting into the behavior that is the default in the Standards mode. Still, you should use the Standards mode—not the Almost Standard mode.

The [HTML specification](http://www.whatwg.org/specs/web-apps/current-work/multipage/) calls this mode the “limited quirks mode”.

### The Mode for application/xhtml+xml Content (XML Mode)

In Firefox, Safari, Chrome, Opera and IE9, the application/xhtml+xml HTTP Content-Type (*not* a meta element nor a doctype!) triggers the XML mode. In the XML mode, these browsers give the specification-wise correct treatment for XML documents to the extent implemented in a particular browser.

IE 6, 7 and 8 did not support application/xhtml+xml. Neither did Mac IE 5.

In the WebKit-based Nokia S60 Browser, the application/xhtml+xml HTTP Content-Type did not trigger the XML mode due to concerns of compatibility with ill-formed content in mobile walled gardens. ([Legacy “mobile browsers” didn’t use a real XML parser](http://simon.html5.org/articles/mobile-results) and, therefore, ill-formed content has been labeled as XML.)

I have not tested the default browser on Symbian3.

I have not tested Konqueror sufficiently to say what exactly happens in that browser.

### IE-Specific Additional Modes from Microsoft

The following are additional IE-specific modes that are not specified by HTML5 and that other browsers don’t have. Their activation involves configuration or X-UA-Compatible as an HTTP header or a meta element ([discussed below](https://hsivonen.fi/doctype/ie8)).

Internet Explorer 5 Quirks

In addition to the interoperable Quirks mode, IE10 also has a mode called “Internet Explorer 5 Quirks”, which imitates IE 5.5 and is the mode that was known as the Quirks mode in IE6, IE7, IE8 and IE9.

Internet Explorer 7 Standards

IE8, IE9 and IE10 have a mode that imitates the mode that was the Standards mode in IE7.

Internet Explorer 8 Standards

IE9 and IE10 have a mode that imitates the mode that was the Standards mode in IE8.

Internet Explorer 8 Almost Standards

IE9 and IE10 have a mode that imitates the mode that was the Almost Standards mode in IE8. In the developer tool user interface, this mode is not distinguished from “Internet Explorer 8 Standards”.

Internet Explorer 9 Standards

IE10 has a mode that imitates the mode that was the Standards mode in IE9.

Internet Explorer 9 Almost Standards

IE10 has a mode that imitates the mode that was the Almost Standards mode in IE9. In the developer tool user interface, this mode is not distinguished from “Internet Explorer 9 Standards”.

Internet Explorer 9 XML

IE10 has a mode that imitates the mode that was the XML mode in IE9. In the developer tool user interface, this mode is not distinguished from “Internet Explorer 9 Standards”.

It is worth noting that the imitations of the previous versions of IE are not perfect. Random examples that I have encountered myself include IE7 Standards emulation in a later IE version handling @font-face-liked EOT fonts differently and the IE9 modes of IE10 supporting CSS 2D Transformations without the -ms- when the real IE9 requires the prefix. If you follow the advice given in this document, you won’t be targeting these modes, so the imperfections of the imitations won’t matter to you in production. However, the takeaway for testing is that you’ll be better off testing your site in actual old versions of IE running in a bunch of virtual machines than using the developer tools newer IE versions to make the new version emulate an old version for testing.

IE10 for Windows Phone 8 has all these modes, too—just like IE10 on desktop.

### IE-Specific Additional Modes from Google

The following are additional modes available in IE6, IE7, IE8 and IE9 (but not in IE10 on Windows 8 or on Windows 7 as of February 2013) when [Google Chrome Frame](https://developers.google.com/chrome/chrome-frame/) has been installed.

Chrome Quirks

This mode is the same as the Quirks mode in Google Chrome.

Chrome Standards

This mode is the same as the Standards mode in Google Chrome.

Chrome Almost Standards

This mode is the same as the Almost Standards mode in Google Chrome.

### Non-Web Modes

Some engines have modes that are not relevant to Web content. These modes are only mentioned here for completeness. Opera has a WML 2.0 mode. WebKit on Mac OS X 10.5 had a special mode for legacy Dashboard widgets (perhaps this mode remains in newer versions—I have not investigated). WebKit also has hacks for applications that embed WebKit on Mac OS X.

## The Effects

Here are the main effects of the modes:

### Layout

Except in IE, *the modes for text/html mainly affect CSS layout and the style system*. For example, not inheriting styles into tables is a quirk. In old versions of IE and Opera, the box model changes to the IE 5.5 box model in the Quirks mode. This document does not enumerate all the layout quirks. For a list, please refer to [Mozilla’s documentation](https://developer.mozilla.org/en-US/docs/Mozilla_Quirks_Mode_Behavior) and the [Quirks Mode specification](http://quirks.spec.whatwg.org/).

In the Almost Standards mode (in browsers that have one and all the current ones do), the height of table cells containing only images is computed differently compared to the Standards mode.

In the XML mode, selectors have different case-sensitivity behavior. Furthermore, special rules for the HTML bodyelement do not apply in older versions of browsers that do not implement the more recent adjustments to the CSS specifications.

### Parsing

There are also some quirks that affect HTML and CSS parsing and would cause conforming pages to be misparsed. These quirks are toggled on and off with quirky layout. However, it is important to realize that the Quirks mode vs. Standards mode is predominantly about CSS layout and CSS parsing—not HTML parsing. In browsers that have an HTML5-compliant HTML parser, there is [exactly one HTML parsing quirk](http://hsivonen.iki.fi/last-html-quirk/).

Some people misleadingly refer to the Standards mode as “strict parsing mode”, which is misunderstood to imply that browsers enforced HTML syntax rules and that a browser could be used to assess the correctness of markup. This is *not*the case. The browsers do tag soup fix-ups even when the Standards mode layout is in effect. (In the summer of 2000 before Netscape 6 was released, Gecko actually had parser modes that enforced HTML syntax rules and one of these modes was called the “Strict DTD”. These modes were incompatible with existing Web content and were abandoned.)

Another common misconception is related to XHTML parsing. It is often thought using an XHTML doctype gains different parsing. *It does not.* XHTML documents served as text/html are parsed using the *same* parser that is used for HTML. As far as browsers are concerned, XHTML served as text/html is just “tag soup with croutons” (an extra slash here and there). Only documents served using an XML content type (e.g. application/xhtml+xml or application/xml) trigger the XML mode for parsing in which case the parser is totally different from the HTML parser.

### Scripting

Although the Quirks mode is primarily about CSS, there are some scripting quirks as well. Until Firefox 14, the HTML idattribute didn’t establish object references from the global scripting scope in the Standards and Almost Standards modes. In Firefox, document.all is partially available in the Quirks mode but not in the other modes. The effects on scripting are more dramatic in IE when falling into a mode that emulates an old version of IE.

In the XML mode, some DOM APIs behave differently, because the DOM API behavior for XML is defined to be incompatible with the HTML behavior. With hindsight, this is quite unfortunate.

## Doctype Sniffing (aka. Doctype Switching)

Browsers use doctype sniffing in order to decide the engine mode for text/html documents. This means that the mode is picked based on the document type declaration (or the lack thereof) at the beginning of an HTML document. (This does not apply to documents served with an XML content type.)

A document type declaration (doctype) is a syntactic artifact of SGML—a legacy markup framework that HTML prior to HTML5 was purportedly defined in terms of. In the HTML 4.01 specification, the document type declaration is said to communicate HTML [version information](http://www.w3.org/TR/html4/struct/global.html#h-7.2). Despite the name “document type declaration” and despite what the HTML 4.01 specification says about “version information”, *the document type declaration is not an appropriate means for classifying SGML or XML documents as documents of a particular type* even though it seems it was supposed to be (hence the name). (More on this [in the addendum](https://hsivonen.fi/doctype/#xml).)

Neither the HTML 4.01 specification nor ISO 8879 (SGML) says anything about using the document type declaration as an engine mode switch. Doctype sniffing is based on the observation that at the time doctype sniffing was devised the vast majority of quirky documents either didn’t have a document type declaration or they referenced an old DTD. HTML5 acknowledges this reality and defines the doctype in text/html as a mode switch only.

A *typical* pre-HTML5 document type declaration contains (separated by white space) the string “<!DOCTYPE”, the generic identifier of the root element (“html”), the string “PUBLIC”, a public identifier of a DTD in quotes, possibly a system identifier (an URL) of the same DTD and the character “>”. HTML5 simplies the doctype to “<!DOCTYPE html>”. The document type declaration is placed in the document before the start tag of the root element.

## Choosing a Doctype

### text/html

Here are simple guidelines for choosing a doctype for a new text/html document:

Standards mode, cutting edge validation

<!DOCTYPE html>

This is what you should use. With this doctype, you can [validate](http://html5.validator.nu/) new features such as <video>, <canvas> and ARIA. Please be sure to test your page in the latest versions of the top browsers.

Standards mode, legacy validation target

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

This doctype also triggers the Standards mode, but lets you stick to less precise legacy validation that doesn’t know about new features in case your organization has silly policies that require targeting legacy validation. But you really should be using <!DOCTYPE html> and get the policies of your organization revised.

You’d like to use the Standards mode, but you use sliced images in table layouts and don’t want to fix them

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

This gives you the Almost Standards mode. Please note that your layouts based on sliced images in tables are likely to break if you later move to HTML5 (and, hence, the full Standards mode), so it’s better to make your designs Standards mode-compatible right now.

You willfully want the Quirks mode

No doctype.

Please don’t do this. Willfully designing for the Quirks mode will come and haunt you, your coworkers or your successors in the future.

If you frustrated by the differences between old IE versions and still need to support them due to client requirements, it is better to apply specific hacks for legacy versions using [conditional comments](http://www.quirksmode.org/css/condcom.html) than seek commonality in the Quirks mode.

I am not recommending any of the XHTML doctypes, because [serving XHTML as text/html is considered harmful](http://hixie.ch/advocacy/xhtml). If you choose to use an XHTML doctype anyway, please note that the XML declaration makes IE 6 (but not IE 7!) trigger the Quirks mode.

### application/xhtml+xml

The simple guideline for application/xhtml+xml is *not* to use a doctype *at all*. This way the page cannot be “strictly conforming” XHTML 1.0, but that does not matter. (Please see the [Addendum](https://hsivonen.fi/doctype/#xml) below.)

## IE8, IE9 and IE10 Complications

It was [announced on A List Apart](http://www.alistapart.com/articles/beyonddoctype) that IE8 would use a meta element-based mode switch in addition to doctype being a factor in the mode choice. (See commentary by [Ian Hickson](http://ln.hixie.ch/?start=1201080691&count=1), [David Baron](http://lists.w3.org/Archives/Public/public-html/2007Apr/0279.html), [David Baron again](http://dbaron.org/log/2008-01#e20080124a), [Robert O’Callahan](http://weblogs.mozillazine.org/roc/archives/2008/01/post_2.html) and[Maciej Stachowiak](http://webkit.org/blog/155/versioning-compatibility-and-standards/).)

IE8 has four modes: IE 5.5 quirks mode, IE 7 standards mode, IE 8 almost standards mode and IE 8 standards mode. IE9 has *seven* modes: IE 5.5 quirks mode, IE 7 standards mode, IE 8 almost standards mode, IE 8 standards mode, IE 9 almost standards mode, IE 9 standards mode and IE 9 XML mode. IE10 has *eleven* modes: IE 5.5 quirks mode, IE 7 standards mode, IE 8 almost standards mode, IE 8 standards mode, IE 9 almost standards mode, IE 9 standards mode, IE 9 XML mode, IE 10 quirks mode, IE 10 almost standards mode, IE 10 standards mode and IE 10 XML mode. The choice of mode depends on data from various sources: doctype, a meta element, an HTTP header, periodically downloaded data from Microsoft, the intranet zone, settings made by the user, settings made by an intranet administrator, the mode of the frame parent if any and a UI button togglable by the user. (With other apps that embed the engine, the mode also depends on the embedding application.)

The lucky thing is that IE8 and IE9 use doctype sniffing roughly like other browsers and IE10 uses doctype sniffing exactly like other browsers if all the following points are true:

* There is no X-UA-Compatible HTTP header set by the author.
* There is no X-UA-Compatible meta tag set by the author.
* Microsoft has not placed the *domain name* of the site on a [blacklist](http://go.microsoft.com/fwlink/?LinkId=145413).
* An intranet admin has not placed the site on a blacklist.
* The user has not pressed the Compatibility View button (or otherwise added the domain to a user-specific blacklist). (Metro IE10 does not have this UI, but the UI in desktop IE10 affects the behavior in the Metro mode also.)
* The site is not in the intranet zone.
* The user has not chosen to display all sites as in IE7.
* The page is not framed by a Compatibility Mode page.

For the points other than the two X-UA-Compatible cases, IE8 and IE9 perform doctype sniffing like IE7. The IE7 emulation is called Compatibility View.

In the X-UA-Compatible cases, IE8 and IE9 behave radically differently from other browsers. For the behavior of IE8, please see [an appendix on this page](https://hsivonen.fi/doctype/#ie8modes) or a flowchart available in [PDF](https://hsivonen.fi/doctype/ie8-mode.pdf) and [PNG](https://hsivonen.fi/doctype/ie8-mode.png) formats. (Contrast with the chart for other browsers as [PDF](https://hsivonen.fi/doctype/html5-mode.pdf).) There is also a unified chart of IE 5.5 through 9 (potentially with Chrome Frame) modes as [PDF](https://hsivonen.fi/doctype/ie-mode.pdf). (I intend to make an IE9-only view of this chart later.)

Unfortunately, without an X-UA-Compatible HTTP header or meta tag, IE8 and IE9 let the user accidentally drop your page from their most standards mode to the IE7 mode that emulates the standards mode of IE7 even if you used a proper doctype. Worse, an intranet admin may do this. Also, Microsoft may have blacklisted the entire domain you use (e.g. mit.edu!).

To counter these effects, a doctype isn’t enough and you need an X-UA-Compatible HTTP header or meta tag.

Here are simple guidelines for choosing an X-UA-Compatible HTTP header or meta tag for a new text/htmldocument that *already has a doctype that triggers the standards mode or almost standards mode in other browsers*:

Your domain is not on Microsoft’s blacklist and you care more about not having to have browser-specific cruft than about making sure users can’t regress the rendering to IE7 behavior

You don’t need to include an X-UA-Compatible HTTP header or meta tag.

Your domain is on Microsoft’s blacklist, your domain (like iki.fi!) has other authors whose broken sites may induce users to enable Compatibility View for the *whole domain*, you are concerned about Google or Digg framing your site or you want to make sure users cannot enable the Compatibility View

Include either the following meta element (which in invalid in HTML5) on your page <meta http-equiv="X-UA-Compatible" content="IE=Edge"> (before any script elements!) or set the following HTTP header on your page: X-UA-Compatible: IE=Edge

Your site worked in IE7 but breaks in IE8 or IE9

First, include either the following meta element (which in invalid in HTML5) on your page <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7"> (before any script elements!) or set the following HTTP header on your page: X-UA-Compatible: IE=EmulateIE7

Then fix your site not to rely on non-standard IE7 behaviors and migrate to IE=Edge.

Your site worked in IE8 but breaks in IE9

First, include either the following meta element (which in invalid in HTML5) on your page <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8"> (before any script elements!) or set the following HTTP header on your page: X-UA-Compatible: IE=EmulateIE8

Then fix your site not to rely on non-standard IE8 behaviors and migrate to IE=Edge.

Your site worked in IE9 but breaks in IE10

First, include either the following meta element (which in invalid in HTML5) on your page <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE9"> (before any script elements!) or set the following HTTP header on your page: X-UA-Compatible: IE=EmulateIE9

Then fix your site not to rely on non-standard IE8 behaviors and migrate to IE=Edge.

## Google Chrome Frame Complications

[Google Chrome Frame](https://developers.google.com/chrome/chrome-frame/) is a combination of a browser extension and browser plug-in for IE 6, 7, 8 and 9 that adds the engine of Google Chrome into the user interface shell of IE using the networking stack that IE uses. After installation, IE behaves normally by default. However, Web pages can opt to invoke the engine of Chrome instead of the engine of IE using an X-UA-Compatible HTTP header or meta tag.

Specifying chrome=1 in X-UA-Compatible invokes Chrome Frame in any supported version of IE if Chrome Frame is installed. Specifying chrome=IE6 activates Chrome Frame in IE6 only, specifying chrome=IE7 activates Chrome Frame only in IE7 and IE6, and chrome=IE8 activates Chrome Frame only in IE8 and lower.

The directive for activating Chrome Frame can be combined with the directives for controlling the engine of IE (in case Chrome Frame is not installed) by separating them with a comma or a semicolon: <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=IE8">.

Once the Chrome Frame has been activated for a page, one of the four modes (Standards, Almost Standards, Quirks and XML) of Chrome is chosen as in normal Chrome.

There are a couple of important reasons against using Chrome Frame:

* Chrome Frame lacks IE’s accessibility support. When Chrome Frame is activated, the content area in IE becomes an accessibility black hole. That is, screen readers and Windows Speech Recognition don’t work with Chrome Frame.
* Making your site tell users that they should install Chrome Frame perpetuates the security anti-pattern of sites telling users that they should let someone install a privileged native-code plug-in on their computer in order to use a site.

## Links to Related Pages

* Eric Meyer writes about the modes of Mac IE 5 in [*Use the Right Doctype*](http://www.oreillynet.com/pub/a/network/2000/04/14/doctype/index.html)
* [*Mozilla’s DOCTYPE sniffing*](http://www.mozilla.org/docs/web-developer/quirks/doctypes.html) by David Baron
* [*CSS Enhancements in Internet Explorer 6*](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnie60/html/cssenhancements.asp) by Lance Silver discusses the modes and doctype sniffing in Windows IE 6
* [*The Opera 9 DOCTYPE Switches*](http://www.opera.com/docs/specs/doctype/)
* Faruk Ateş writes about IE8 in [*IE8 and the X-UA-Compatible situation*](http://farukat.es/journal/2009/05/245-ie8-and-the-x-ua-compatible-situation)

## Addendum: A Plea to Implementors and Spec Writers Working with XML

Please don’t bring doctype sniffing to XML.

Doctype sniffing is a tag soup solution to a tag soup problem. Doctype sniffing was devised *after* the HTML 4 and CSS2 specs had been written as a *heuristic* way to distinguish legacy documents from documents whose authors might expect conforming behavior.

Sometimes it is suggested that doctype sniffing be used on the XML side as well for dispatching to different handlers, for recognizing the vocabulary in use, or for activating features. This is a bad idea. Dispatching and vocabulary recognition should be based on namespaces and feature activation should be based on explicit processing instructions or elements.

The whole concept of well-formedness was introduced to allow DTDless parsing of XML and, by extension, doctypeless documents. In formal terms, if two XML documents have the same canonical form and an app treats them differently (and the difference is not due to opting not to process external entities), chances are the app is broken. In practical terms, if two XML documents cause the same content to be reported (qnames ignored) to SAX2 [ContentHandler](http://www.saxproject.org/apidoc/org/xml/sax/ContentHandler.html)and an app treats the documents differently, chances are the app is broken. Considering that as a Web author you cannot trust that everyone parsing your pages uses an XML processor that resolves external entities (even if some browsers appear to do so because they map certain public ids to an abridged DTD for entity definitions), inserting a doctype in XML intended for the Web is mostly pointless and often done out of a cargo cultish habit. (You can still validate against a DTD using the [DTD override feature](http://validator.w3.org/docs/users.html#option-doctype) of the W3C Validator, although the W3C Validator will say that the result is only tentatively valid. Or better yet, you can use [RELAX NG validation](http://validator.nu/), which does not pollute the document with schema references.) Requiring a doctype only for sniffing would be silly, even though that is the situation with HTML in practice.

Moreover, when a lower-level spec defines two things that are equivalent, a higher-level spec should not try to give different meanings to the two things. Consider <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">. If the public identifier is removed, the same DTD is still designated and, therefore, the doctype <!DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> means the same as the previous doctype. Should they be sniffed differently? The idea can be carried further. Suppose the DTD is copied to example.com and named foobar.dtd: <!DOCTYPE html SYSTEM "http://example.com/foobar.dtd">. How could that one be sniffed? The meaning is the same. The whole DTD could even be pasted inline!

To put it another way, if you have #include "foo.h", you should not bind any black magic to the name foo.h, because it should be permissible to paste the contents of foo.h inline or copy the contents of foo.h to bar.h and say#include "bar.h".

The reason I don’t bother making the same argument with HTML and SGML is that Web browsers don’t use real SGML parsers for parsing HTML, so I don’t think it is useful to pretend that SGML is being dealt with. However, if you are not convinced yet, please see [W. Eliot Kimber’s comp.text.sgml post about the matter](http://groups.google.com/group/comp.text.sgml/msg/c3e53dee2c152a81).

## Appendix: Handling of Some Doctypes in text/html

In the following table, Quirks Mode, Standards Mode and Almost Standards Mode are denoted by Q, S and A, respectively. When a browser only has two modes, the Standards Mode is marked as “S”, if the line height in table cells works as in Mozilla’s Standards Mode, and as “A”, if the line height in table cells works as in Mozilla’s Almost Standards Mode.

Please note that XHTML served using an XML content type is rendered in the XML mode.

The purpose of this table is *not* to suggest that all the doctypes listed in the table are reasonable choices for new pages. The purpose of this table is to show what data I am basing my recommendations on.

The following shorthand notation is used in the column headers:

NS6

Mozilla 0.6…0.9.4 and Netscape 6.0…6.2.3

Old Moz

Mozilla 0.9.5 through 1.1 alpha and Mozilla 1.0

Moz & Safari & Opera 10 & IE10 & HTML5

Mozilla 1.0.1, Mozilla 1.1 beta and later, Firefox, Netscape 7 and later, Safari 0.9 and later, Opera 10 and later, Chrome, Konqueror 3.5, IE10, the HTML5 specified behavior

Opera 9.0

Opera 9.0…9.20

IE 8, IE 9 & Opera 9.5

IE 8 or IE 9 by default when there is no X-UA-Compatible override nor a Compatibility View override (“A” means the IE8 Almost Standards mode for IE8 and IE9 Almost Standards mode for IE9), Opera 7.5…8.54 and 9.5…9.6

IE 7 & Opera 7.10

Windows IE 7, IE 8 with Compatibility View enabled but without X-UA-Compatible override (in this case “A” means the IE7 mode) and Opera 7.10…7.23

IE 6 & Opera 7.0

Windows IE 6 and Opera 7.0…7.03

Mac IE 5

Mac IE 5.0…5.2.3

Konq 3.2

Konqueror 3.2.2…3.3 (possibly also 3.1…3.2.1; I have not been able to confirm)