Browser detection using the user agent

see all contributors

Serving different Web pages or services to different browsers is usually a bad idea. The Web is meant to be accessible to everyone, regardless of which browser or device they're using. There are ways to develop your web site to progressively enhance itself based on the availability of features rather than by targeting specific browsers.

But browsers and standards are not perfect, and there are still some edge cases where detecting the browser is needed. Using the user agent to detect the browser looks simple, but doing it well is in fact a very hard problem. This document will guide you in doing this as correctly as possible.

> It's worth re-iterating: it's very rarely a good idea to use user agent sniffing. You can almost always find a better, more broadly compatible way to solve your problem!

Considerations before using browser detection

When considering using the user agent string to detect which browser is being used, your first step is to try to avoid it if possible. Start by trying to identify **why** you want to do it.

Are you trying to work around a specific bug in some version of a browser?

Look, or ask, in specialized forums: you're unlikely to be the first to hit this problem. Also experts, or simply people with another point of view, can give you ideas for working around the bug. If the problem seems uncommon, it's worth checking if this bug has been reported to the browser vendor via their bug tracking system (Mozilla; WebKit; Opera). Browser makers do pay attention to bug reports, and the analysis may hint about other work-arounds for the bug.

Are you trying to check for the existence of a specific feature?

Your site needs to use a specific Web feature that some browsers don't yet support, and you want to send those users to an older Web site with fewer features but that you know will work. This is the worst reason to use user agent detection, because odds are eventually all the other browsers will catch up. You should do your best to avoid using user agent sniffing in this scenario, and do feature detection instead.

Do you want to provide different HTML depending on which browser is being used?

This is usually a bad practice, but there are some cases in which this is necessary. In these cases, you should first analyze your situation to be sure it's really necessary. Can you prevent it by adding some non-semantic `<div>` or `<span>` elements? The difficulty of successfully using user agent detection is worth a few disruptions to the purity of your HTML. Also, rethink your design: can you use progressive enhancement or fluid layouts to help remove the need to do this?

Avoiding user agent detection

If you want to try to avoid using user agent detection, there are options in some cases!

Feature detection

Feature detection is where you don't try to figure out which browser is rendering your page, but instead you check to see if the specific feature you need is available. If it's not, you use a fallback. However, never use feature detection in the rare cases when you actually want browser detection, since other browsers may implement the feature in the future, but differently. Bugs caused by this can be insidiously hard to find and fix.

Progressive enhancement

This design technique involves developing your Web site in 'layers', using a bottom-up approach, starting with a simpler layer and improving the capabilities of the site in successive layers, each using more features.

Graceful degradation

This is a top-down approach in which you build the best possible site using all the features you want, then tweak it to make it work on older browsers. This can be harder to do, and less effective, than progressive enhancement, but may be useful in some cases.

Which part of the user agent contains the information you are looking for

As there is no uniformity of the different part of the user agent string, this is the tricky part.

Browser Name

When people say they want "browser detection", often they actually want "rendering engine detection". Do you actually want to detect Firefox, as opposed to SeaMonkey, or Chrome as opposed to Chromium? Or do you actually simply want to see if the browser is using the Gecko or the WebKit rendering engine? If this is what you need, see further down the page.

Most browser set the name and version in the format *BrowserName/VersionNumber*, with the notable exception of Internet Explorer. But as the name is not the only information in a user agent string that is in that format, you can not discover the name of the browser, you can only check if the name you are looking for. But note that some browsers are lying: Chrome for example reports both as Chrome and Safari. So to detect Safari you have to check for the Safari string and the absence of the Chrome string, Chromium often reports itself as Chrome too or Seamonkey sometimes reports itself as Firefox.

Also pay attention not to use a simple regular expression on the BrowserName, user agents also contain strings outside the Keyword/Value syntax. Safari & Chrome contain the string 'like Gecko', for instance.

	Must contain	Must not contain	
Firefox	Firefox/xyz	Seamonkey/xyz	
Seamonkey	Seamonkey/xyz		
Chrome	Chrome/xyz	Chromium/xyz	
Chromium	Chromium/xyz		
Safari	Safari/xyz	Chrome/xyz or Chromium/xyz	Safari gives two version number, one technical in the Safari/xyz token, one user-friendly in a Version/xyz token
Opera	OPR/xyz [1] Opera/xyz [2]		[1] Opera 15+ (Blink-based engine) [2] Opera 12- (Presto-based engine)
Internet Explorer	; MSIE xyz;		Internet Explorer doesn't put its name in the *BrowserName/VersionNumber* format

Of course, there is absolutely no guarantee that another browser will not hijack some of these things (like Chrome hijacked the Safari string in the past). That's why browser detection using the user agent string is unreliable and should be done only with the check of version number (hijacking of past versions is less likely).

Browser version

The browser version is often, but not always, put in the value part of the *BrowserName/VersionNumber* token in the User Agent String. This is of course not the case for Internet Explorer (which puts the version number right after the MSIE token), and for Opera after version 10, which has added a Version/*VersionNumber* token.

Here again, be sure to take the right token for the browser you are looking for, as there is no guarantee that others will contain a valid number.

Rendering engine

As seen earlier, in most cases, looking for the rendering engine is a better way to go. This will help to not exclude lesser known browsers. Browsers sharing a common rendering engine will display a page in the same way: it is often a fair assumption that what will work in one will work in the other.

There are five major rendering engines: Trident, Gecko, Presto, Blink and WebKit. As sniffing the rendering engines names is common, a lot of user agents added other rendering names to trigger detection. It is therefore important to pay attention not to trigger false-positives when detecting the rendering engine.

	Must contain	
Gecko	Gecko/xyz	
WebKit	AppleWebKit/xyz	Pay attention, WebKit browsers add a 'like Gecko' string that may trigger false positive for Gecko if the detection is not careful.
Presto	Opera/xyz	**Note:** Presto is no longer used in Opera browser builds >= version 15 (see 'Blink')
Trident	Trident/xyz	Internet Explorer put this token in the *comment* part of the User Agent String
Blink	Chrome/xyz	

Rendering engine version

Most rendering engine put the version number in the *RenderingEngine/VersionNumber* token, with the notable exception of Gecko. Gecko puts the Gecko version number in the comment part of the User Agent after the `rv:` string. From Gecko 14 for the mobile version and Gecko 17 for the desktop version, it also puts this value in the `Gecko/version` token (previous version put there the build date, then a fixed date called the GeckoTrail).

OS

The Operating System is given in most User Agent strings (although not web-focussed platforms like Firefox OS), but the format varies a lot. It is a fixed string between two semi-colons, in the comment part of the User Agent. These strings are specific for each browsers. They indicates the OS, but also often its version and information on the relying hardware (32 or 64 bits, or Intel/PPC for Mac).

Like in all cases, these strings may change in the future, one should use them only in conjunction for the detection of already released browsers. A technological survey must be in place to adapt the script when new browser versions are coming out.

Mobile, Tablet or Desktop

The most common reason to perform user agent sniffing is to determine which type of device the browser runs on. The goal is to serve different HTML to different device types.

- Never assume that a browser or a rendering engine only runs on one type of device. Especially don't make different defaults for different browsers or rendering engines.
- Never use the OS token to define if a browser is on mobile, tablet or desktop. The OS may run on more than one type of (for example, Android runs on tablets as well as phones).

The following table summarizes the way major browser vendors indicate that their browsers are running on a mobile device:

Common browsers User Agent strings

Browser	Rule	Example
Mozilla (Gecko, Firefox)	**Mobile** or **Tablet** token in the comment.	Mozilla/5.0 (Android; Mobile; rv:13.0) Gecko/13.0 Firefox/13.0
WebKit-based (Android, Safari)	**Mobile Safari** token outside the comment.	Mozilla/5.0 (Linux; U; Android 4.0.3; de-ch; HTC Sensation Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Blink-based (Chromium, Google Chrome, Opera 15+)	**Mobile Safari** token outside the comment	Mozilla/5.0 (Linux; Android 4.4.2); Nexus 5 Build/KOT49H) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.117 Mobile Safari/537.36 OPR/20.0.1396.72047
Presto-based (Opera 12-)	**Opera Mobi/xyz** token in the comment (Opera 12-)	Opera/9.80 (Android 2.3.3; Linux; Opera Mobi/ADR-1111101157; U; es-ES) Presto/2.9.201 Version/11.50
Internet Explorer	**IEMobile/xyz** token in the comment.	Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0)

In summary, we recommend looking for the string “Mobi” anywhere in the User Agent to detect a mobile device.

> If the device is large enough that it's not marked with “Mobi”, you should serve your desktop site (which, as a best practice, should support touch input anyway, as more desktop machines are appearing with touchscreens).