



A BRIEF HISTORY OF JAVASCRIPT

By Jeremy Keith

Web standards are on the rise. The new standards-based approach to building websites is fundamentally different to the old-fashioned techniques of using tables and single-pixel transparent .gifs for layout. Instead, clued-up web designers are approaching documents on the Web in terms of layers of enhancement. Begin with well marked-up content, then add CSS for presentation and DOM Scripting for behaviour. This separation of structure, presentation and behaviour is a central tenet of standards-based web design.

The presentation layer has been the focus of most activity in recent years. CSS has really taken off which, in some ways, is surprising. Don't get me wrong - CSS is a marvelous technology. The problem lies with

browser support. There is a considerable gulf between the specifications for CSS and the implementation of those specs in the most popular web browsers.

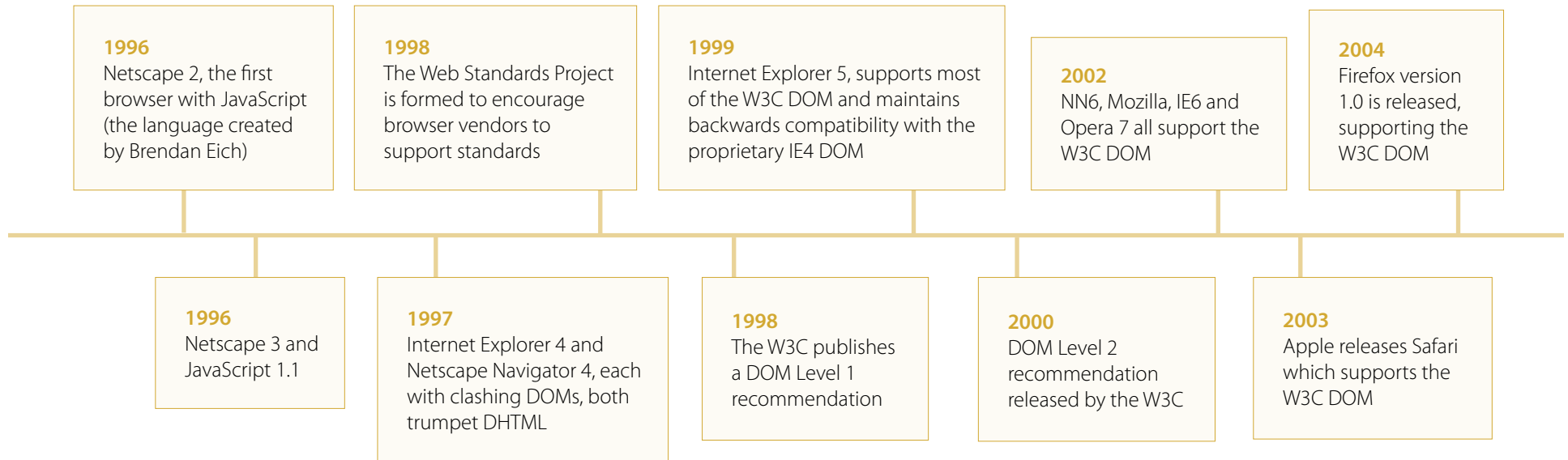
Compared to CSS, DOM Scripting has been largely ignored. This is also surprising given that the gulf between the specs and the implementations are very narrow indeed. But it wasn't always like this.

The very first web browser, created by [Tim Berners-Lee](#), was a text-only affair. Later, Marc Andreessen created Mosaic, which threw images into the mix. The next big step in browser evolution was the introduction of the first version of JavaScript in [Netscape Navigator 2](#). The year was 1995.

JavaScript was introduced to add extra "oomph!" to web pages. Instructions written in JavaScript were interpreted and executed by the web browser. Most of these instructions involved the manipulation of the browser window: changing its size, moving it around and spawning new windows. These dubious actions were possible because JavaScript could communicate directly with the browser using a sort of Browser Object Model.

In 1996, [JavaScript 1.1](#) appeared in version 3 of both Netscape's Navigator and Microsoft's Internet Explorer, adding more power to the language. It became possible to manipulate certain elements of the web document within the browser window - mostly images and forms. This was an early form of a Document Object Model - a way of "talking to" a document in

JavaScript Timeline



order to manipulate its contents.

When Netscape and Microsoft released version 4 of their browsers, they promised a world of possibilities. But what they delivered was a crushing disappointment.

A new buzzword was coined: DHTML. In theory, Dynamic HTML would allow developers to create richer web pages filled with movement and interactivity. This would be achieved by using a new, expanded DOM to update the styles of a page element on the fly.

There were two problems with DHTML (apart from its confusing name). Firstly, CSS support was lousy in both browsers. Secondly, each browser used its own proprietary Document Object Model.

Suppose you wanted to find a <div> with the ID “bar” and assign it to a variable, foo.

In NN4, you had to write:

```
var foo = document.layers["bar"];
```

IE4 meanwhile, required the following syntax:

```
var foo = document.all["bar"];
```

This thoroughly annoying state of affairs meant that any time you wanted to add even the simplest bit of DHTML, you needed to fork your code. To add insult to injury, this balkanisation of JavaScript was completely unnecessary. A standardised DOM already existed. It is a specification from the W3C and its syntax looks like this:

```
var foo = document.getElementById("bar");
```

At first glance, this appears to be little better than the proprietary DOMs. However, the W3C DOM goes much, much further. It allows you to manipulate any node in a document. That means that as well as manipulating elements, you can also create, modify and destroy attributes and text. It isn't limited to HTML either.

“The Document Object Model is a platform and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.”

What is the Document Object Model?

The DOM is an API, an Application Programming Interface. It can be used by any language to manipulate any kind of markup. The DOM Scripting skills that you pick up when you're using JavaScript to manipulate a web page could come in handy when you're using PHP to parse an XML file.

The refusal by Netscape and Microsoft to implement the standardised DOM was one

indignity too many in the already vicious browser wars of the late nineties. The browser makers' disregard for the W3C DOM led directly to the creation of the Web Standards Project.

Whether through the efforts of the WaSP or as a result of internal soul searching, Netscape and Microsoft began to implement the W3C's recommendations. But the damage was already done.

The term DHTML became a byword for difficult, buggy cross-browser hacking. By the time the next browser revisions arrived, web developers were no longer interested in the DOM or JavaScript. We turned our attention to CSS and we turned our backs on DOM Scripting.

And here's the curious thing: while our backs were turned, the dream of cross-browser DOM support became a reality. Sure, there are still inconsistencies from browser to browser, but for the most part, nearly every major browser supports the core DOM functionality. Compare that to the situation with CSS. Years after the technology was first implemented, browser support is still the number one headache for any CSS developer.

If you've been ignoring JavaScript and the Document Object Model for the past few years, perhaps it's time for a re-evaluation.

JavaScript has a steeper initial learning curve than CSS or HTML (it is, after all, a programming language), but once you've got something working in one browser, you may be pleasantly surprised when you test it in other browser/platform combinations. More often than not, you can code once and deploy everywhere - a far cry from the code-forking days of the browser wars.



Jeremy Keith is the author of DOM Scripting: Web Design with JavaScript and the Document Object Model and serves as joint leader of the WaSP's DOM Scripting Task Force. He works in Brighton, England for the web consultancy firm Clearleft. His online home is adactio.com. When he's not building web sites, Jeremy plays bouzouki in the alt.country band Salter Cane.