

The Document Object Model: an Introduction

January 24, 2014 by digiwebs

The Document Object Model: an Introduction

Got something to say?

[Share your comments](#) on this topic with other web professionals

In: [Articles](#)

By [Peter-Paul Koch](#)

Published on May 14, 2001

In recent months much has been said about the Javascript DOM, the Document Object Model. It was rumored that Netscape 6 has a new one, and that it's really important to know how to use it. Nonetheless some people are a bit hazy about what a DOM is, and why they should want to use it. In this article I'd like to give a short introduction and history of the DOM. It's not really a tutorial, but after you've read this you'll hopefully have a better grasp on this enigmatic model and why it is important to web development.

First, the good news: if you've ever done anything with JavaScript, even copy-and-pasting a little mouseover script into your personal page, you have already used the DOM. If the script worked, you used the DOM correctly. So it isn't that hard to use, the only problem is that there are no less than four DOMs around at the moment and if you really want to learn JavaScript you'll have to know them all. Don't worry, I'll explain each of the four below.

In this article I won't give too many coding examples. For the gory details of making the DOMs work, please see my [JavaScript Section](#).

Definition of a Document Object Model

A Document Object Model is a model of how the various HTML elements in a page (paragraphs, images, form fields, etc.) are related to each other and to the topmost structure: the document itself. So the document is represented as a kind of tree, in which each HTML element is a branch or leaf, and has a name.

I see the use of the DOM as a kind of naming magic. If you call on an HTML element using its proper name, you are granted access and you can influence the element, forcing the browser to react to your arcane incantations. Of course, like in fairy tales, if you use a wrong name or try to influence the wrong property, terrible things may start to happen.

Therefore it is very important that you know the proper incantations (plural, because sometimes you need to know several names for the same element).

For instance, when you write a rollover script you access a certain image in the page by using its correct name:

```
document.images[ 'thename' ]
```

When you are granted access, you can change its src property. As soon as you do that, the browser reacts to your "spell" by loading another image in the place of the first. If the image you try to name doesn't exist, however, or if you misspelled the name, the browser gives error messages and your magic won't work.

As you might have guessed, older browsers only give access to a limited number of HTML elements, while the newest browsers give access to everything. Also, you can change only a few things in older browsers, while you can change pretty much anything in the newest browsers. Therefore it's always important to know if the various browsers can do what you want them to do. If you give them orders they can't execute, they start sulking and give JavaScript Error Message Alerts.

Level 0 DOM

The first DOM, the Level 0 DOM, was invented by [Netscape](#) when JavaScript as a whole was invented. The idea behind JavaScript was to give web developers means by which a site could genuinely interact with its user. Whatever the concept behind your pages, interaction always means changing things in a web page in response to user-generated events. To change an HTML element you need access to it, hence a DOM is a requirement for JavaScript.

Netscape understood this need quite well, devised the Level 0 DOM, and built it into Netscape 2. It was still very simple, you could only access forms, links and (in Netscape 3) images, but web developers were enthusiastic about it. They could check what people had filled in in forms! They could create the famous rollover effect! It seemed like living in Paradise.

Of course, in due time a snake showed up in the glade: Explorer 3.0. [Microsoft](#) wanted to gain a foothold on the browser market and since Netscape called the shots back then, Microsoft had to adjust. Microsoft wanted to create a true competitor for Netscape 3, so having Explorer 3 produces lots of error messages on each page containing JavaScript would have been strategically unsound.

Therefore Microsoft also started using the Level 0 DOM and the same names gave access to the same elements in Netscape 3 and Explorer 3.

But not quite. People quickly found out that Explorer 3 didn't give access to the images on a page, so that the mouseovers wouldn't work. Even worse, when you tried calling an image by its proper DOM name, Explorer 3 would produce errors because it didn't understand what you were talking about. So web developers were forced to take compatibility questions into account. Don't start calling `document.images` immediately – check first, to ensure that it's supported by that visitor's browser. This was the beginning of support detection:

```
if (document.images) { // do something with document.images, for instance: document.images['t'
```

...So first check if the browser supports `document.images` at all, and only when it does, call the image by its proper name and change its properties.

Intermediate DOMs

Though web developers thought they had a pretty tough time working around the minor incompatibilities between Netscape 3 and Explorer 3, this was nothing compared to the toil and trouble the Version 4 browsers brought with them. The buzzword of the day was DHTML: influencing style sheets by means of JavaScript methods.

DHTML was supposed to give web developers the opportunity of changing a web page on the fly, for instance by adjusting the position of a layer. Since more HTML elements needed to be accessible, the DOM had to be extended. In view of their increasing competition it is not surprising that Netscape and Microsoft decided to implement their own proprietary DOMs, `document.layers` for Netscape and `document.all` for Explorer. These were the two Intermediate DOMs.

The Intermediate DOMs offered access to what are popularly known as layers: independent parts of the page that could be moved or hidden¹. In addition, the Explorer 4 DOM also offered access to most other HTML elements (paragraphs, `<td>`'s), though actually changing the properties of these elements sometimes didn't work quite properly.

Web developers groaned and moaned and wrote more complicated scripts to make sure both browsers could handle their DHTML. For instance, to adjust the position of the layer with `id="layername"`:

```
if (document.layers) { document.layers['layername'].top = 200; } else if (document.all) { document.
```

The `document.layers` bit was executed in Netscape 4, the `document.all` bit in Explorer 4. So far so bad – the browser-specific coding was not what developers had in mind for writing simple web pages, but it could be handled.

A worse problem was that Netscape 4 offered far less access than Explorer 4. In Explorer you could change the colour or the margin of a paragraph, in Netscape you couldn't. This difference was partly balanced by the fact that Netscape was released slightly earlier and had far better and more accessible documentation.

On the other hand Netscape's DOM was far more complex than Microsoft's. Netscape insisted on making each layer a separate document, so that if you want to access an image inside a layer you'd have to write your code like this:

```
document.layers['layername'].document.images['imagename']
```

The image is inside the document that's inside the layer. Although it isn't entirely illogical this model quickly becomes too verbose. In contrast, for Explorer you could still use the familiar

```
document.images['imagename']
```

reference, because Explorer didn't put separate documents inside the layer. Therefore, the Microsoft DOM was easier to learn and use.

For reasons of backward compatibility, the Version 4 browsers still supported the Level 0 DOM, so that the old form validation scripts and mouseovers still functioned. The number of DOMs now had reached three, the old Level 0 DOM for the old effects and the two Intermediate DOMs for DHTML.

Level 1 DOM

Meanwhile the [World Wide Web Consortium](https://www.w3.org/) had started working on the specifications for the XML DOM, also called the Level 1 DOM. The objective of the new DOM was to provide access to each and every part of an XML document, including comments and processing instructions. It was meant to work for any programming language that could parse and manipulate XML documents.

Since an HTML document can be parsed like an XML document, the new DOM would also be made accessible to JavaScript for the purpose of creating a standard by which the entire tree of a Web document could be accessed.

This standard was adopted by Microsoft and the Mozilla Project (the development team that developed Netscape 6.x) as a result of developer support mobilized by the [Web Standards Project](#) from 1998 onward.

There are other browsers that provide support for this standard as well, most notably [Opera](#) and [Konqueror](#). However, Opera only supports the subset of the DOM that makes possible simple DHTML effects, while all of the other browsers mentioned have attempted to support the entire standard.

Microsoft has (quite rightly) decided that Explorer 5 should continue to support the document.all DOM, thus providing backward compatibility for the many scripts that were written to work in IE4.. Despite this, the Windows and Macintosh versions of Internet Explorer differ considerably, so you cannot be certain that scripts developed on one platform will work properly on the other.

[The Mozilla Project](#) took a completely different approach with their decision to remove completely the complicated and buggy document.layers DOM. Their reason for doing this was that they were going to rewrite Netscape from scratch anyway – so why build in something that’s horrendously complicated? The drawback is, of course, that the scripts written to work in Netscape 4.x will fail in Netscape 6.x. Netscape 6.x and Mozilla don’t provide native support for the document.all DOM, either.

That’s what all the hubbub is about: you have to rewrite your scripts to make them work in Netscape 6. I don’t think that this is such a bad thing, because it becomes necessary to learn the basics of the W3C DOM if you’re going to write DHTML that works in Netscape. It may seem like something of a trial, but since the Level 1 DOM is (supposed to be) a lasting standard, you can learn it with the confidence that you’ll be acquiring knowledge of lasting value.

The Level 1 DOM is supported at least in part across a wide range of recent browsers, and is comprehensive in its methods for accessing the elements of a Web document. Simple scripts will work without difficulty in all of these browsers, though attempts at more sophisticated effects may be difficult, in part because [certain browser vendors](#) have also added their own proprietary extensions.

As said before, the goal of the Level 1 DOM is to provide access to each part of an XML (or HTML) document. This means there are also methods and properties for reading out and even changing the comments in your page. Although this may be quite useful when editing XML documents, I don’t think web developers would be much interested in this functionality. I also doubt that DocumentFragments, NamedNodeMaps and ProcessingInstructions will play a significant role in web development.

In a way this is fortunate. To start using the new DOM you only have to know a few simple things, and when you want to write very complex scripts with lots of browser incompatibilities you can turn to my [compatibility](#) table and look up the specific things you need.

The document tree

In the Level 1 DOM all HTML elements are part of the document tree. This tree starts with the document itself and then goes down to the level of individual p's and b's and br's. Take this example document

```
<html> <head> <title>An example of a document tree</title> </head> <body> <h3>The docum
```

The document has two children, head and body. head has one child, title, while body has four children: one h3 and three p's. In addition, the title, the h3 and two of the p's have one child: the text node that contains the actual text. The second p has even three children: one text node, then a br, then another text node.

You can walk through the entire tree, saying, for instance, "Go to the child of fourth child of the second child of the document and change its value to 'Foo-Bar'":

`document.childNodes[1].childNodes[3].firstChild.nodeValue = 'Foo-Bar'`, which "magically" changes the text of the last paragraph to 'Foo-Bar'. You could even say "Append the same node to the first child of body":

`document.childNodes[1].childNodes[1].appendChild(document.childNodes[1].childNodes[3])` which – in the context of our example document – transfers the entire p into the h3.

However, this code will change the structure of the document tree. If you try to execute the same code again, you'll get an error message. After all, the body doesn't have a fourth child any more, you just moved it to another position. So going through the entire DOM tree is not the best way to access an element. It's far better to use its ID, its unique name.

```
<p id="the_unique_element">It is, like, totally awesome.</p>
```

By giving our p an id we can call it by its name:

`document.getElementById( 'the_unique_element' )` and it will respond, regardless of its location in the document. This makes your elements easier to find, and makes possible a much more robust script.

Conclusion: The promise of DHTML

The promise of DHTML has only really come true now that the Version 5/6 browsers are here. Now you can rewrite your pages on the fly. Do you want to sort a large table by product color instead of product name? No problem. Access the correct td's, read out the values of their text nodes, sort them

alphabetically, completely rewrite the table, and display the new sorting order. No more round-trips to the server are necessary.

If part of your job is to write JavaScript that makes a page interact with the user, I strongly recommend that you establish proficiency with the W3C Level 1 DOM. If you're something of a newcomer to DHTML, or intimidated by the whole idea of learning a new DOM, it might help to visit my [introduction to the Level 1 DOM](#), which features some simple explanations and examples.

Resources

When you're ready for the real work, check out these sites. They'll give you interesting tips and tricks about various aspects of the DOM:

- [Mozilla – Traversing a Table](#). Simple example script that messes with a table.
 - J. David Eisenberg's excellent series of articles in A List Apart:
 - [Meet the DOM](#): About the DOM in general and the differences with the earlier browser specific DOMs.
 - [DOM Design Tricks 1](#): About the *display* style declaration.
 - [DOM Design Tricks 2](#): About event capturing in Netscape 6.
 - [DOM Design Tricks 3](#): About the changing of texts in a document. About nodes.
 - [Working with JavaScript – Modifying Styles](#). Article by Steve Champeon about how to change the styles of entire classes using the `stylesheets` array of the new DOM.
 - [Scottandrew.com](#). Lots of articles about the newest browsers and their failings. Includes an excellent [DOM Introduction](#).
 - [PBWizard](#). Interesting examples of and articles about the W3C DOM and related standards.
-

Footnotes

¹ The term 'layers' was coined by Netscape and it was also the name of its Intermediate DOM. Since in the beginning of DHTML the Netscape model was considered the standard and Microsoft's only a strange extension, the Netscape name has become the standard term.

[Back to content](#)

Got something to say?

Share your comments with other professionals (0 comments)

- [Add to Magnolia](#)
- [Add to My Yahoo!](#)
- [Add to Newsvine](#)
- [Digg this story](#)
- [Add to Del.icio.us](#)

Related Topics: [DOM](#), [Scripting](#)



[Peter-Paul Koch](#) is a freelance web developer, writes and maintains the [Quirksmode.org](#) site. He is also an Administrator of the [WDF](#) and [WDF-DOM](#) mailing lists.

articles

- < [The Destination Matters More Than the Journey](#)
- > [The evolt.org community](#)

Recent Posts



[The Power and Problems of Provisional Personas in UX design](#)

January 19, 2016



[The Best SEO Companies in Texas](#)

January 29, 2026



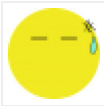
[Best SEO Companies in Austin](#)

January 28, 2026

[Articles by Topic: Search Engine Optimization \(SEO\)](#)August 3, 2018

[Articles by Topic: RSS](#)August 3, 2018

[Articles by Topic: Liquid Web Design](#)August 3, 2018

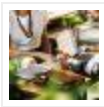
[Articles by Topic: Keep It Simple \(KISS\)](#)August 3, 2018

[Articles by Topic: Information Design](#)August 3, 2018

[Articles by Topic: Graphic Design](#)August 3, 2018

[Articles by Topic: Flash](#)August 3, 2018

[Art of Interaction Column](#)August 3, 2018

[Articles by Topic: Business](#)August 1, 2018

[Web Design 101](#)August 1, 2018

[Book Reviews](#)August 1, 2018

[Articles](#)August 1, 2018

[Articles by Topic: Human-Computer Interaction \(HCI\)](#)August 1, 2018



Digital Web Magazine is a non-profit publication created by a world-wide network of volunteers dedicated to producing quality educational and informative material free-of-charge, to foster the development of the web for the benefit of all.

Email: info@digital-web.com

Recent Posts

[The Best SEO Companies in Texas](#)[Best SEO Companies in Austin](#)[Articles by Topic: Search Engine Optimization \(SEO\)](#)[Articles by Topic: RSS](#)[Articles by Topic: Liquid Web Design](#)

© 2019 Digital Web