PROGRAMMER TO PROGRAMMER ™

e-book includes this complete reference and 2000+ extra entries

JavaScript

Programmer's Reference

Cliff Wootton

JavaScript Programmer's Reference

Cliff Wootton

Wrox Press Ltd. ®

JavaScript Programmer's Reference

Published by Wrox Press Ltd,
Arden House, 1102 Warwick Road, Acocks Green,
Birmingham, B27 6BH, UK
Printed in the United States
ISBN 1-861004-59-1

Trademark Acknowledgements

Credits

Author
Cliff Wootton

Category Manager
Dave Galloway

Technical Editors
Timothy Briggs
Howard Davies
Phillip Jackson
Amanda Kay
Simon Mackie
Chris Mills
Peter Morgan

Project Manager
Chandima Nethisinghe

Production Coordinator
Tom Bartlett

Additional Layout
Simon Hardware
Pippa Wonson

e-Book Production
Tom Bartlett

Production Manager
Simon Hardware

Technical Reviewers
Alex Abacus
Jonny Axelsson
Chong Chang
Andrew Van Heusen
Martin Honnen
Ron Hornbaker
Kenneth Lo
Jim Macintosh
Jon Stephens
Peter Torr
Chris Ullman
Paul Vudmaska
Paul Wilton

Figures
Shabnam Hussain

Cover
Shelley Frazier

Proofreaders
Ian Allen
Christopher Smith
Agnes Wiggers

Index
Andrew Criddle

About the Author

Cliff Wootton lives in the south of England and works on multimedia systems and content management software for large data driven web sites. Currently he is developing interactive TV systems for BBC News Online in London (http://www.bbc.co.uk/news) and previously worked for other commercial broadcasters on their web sites. Before that he spent several years developing geophysical software and drawing maps with computers for oil companies.

Cliff is married with three daughters and a growing collection of bass guitars.

Acknowledgements

It's hard to believe I've actually reached the stage of writing the introductory pages to this book. It's been a long process and I don't think I would have reached this point without the help of Tim Briggs at Wrox, who very gently urged me onwards and gave me encouragement when I needed it. Tim's contribution to this project was vital to its success because he developed the process which converted my DOCBOOK output into something the Wrox editors could turn into a book. Tim also prepared the CD-ROM content from the same XML files; truly amazing!

Thanks also to all the other folks at Wrox who have helped, organised, checked and collated my material to present it in the form you now see it. Grateful thanks to my reviewers, who in a very short time provided me with some useful guidance and support; in particular Jon Stephens and Martin Honnen, who also provided some amazingly clever example code fragments for use as examples.

There are many other people who contributed without realising it. In particular Nick Cohen (formerly of the BBC and now at Turner Broadcasting) who provided some helpful insights into TV set-top-box workings. Also Matt Karas and Emyr Tomos (both ex-BBC, now at Talkcast) who threw down the gauntlet of several interesting challenges for me to implement on the BBC News Online web site. I also wouldn't be sitting here if it weren't for Bruce Morris at Carlton Online. It was through the happy chance of an article I wrote for Bruce's Web Developer's Journal (WDJ) web site that led to Wrox contacting me and the BBC inviting me to do some JavaScript work. What an amazing thing the web is.

Most importantly I dedicate this book to my family. To my wife, Julie and my daughters Hannah, Lydia and Ruth who kept me going with cups of coffee, hugs and the occasional giggle when they saw the photograph of me for the front cover.

Introduction

The JavaScript language is constantly developing, and continues to increase in popularity. Its evolution into a general purpose scripting language from what started life purely for scripting web browsers, is a great success story. You can now find JavaScript interpreters in many different environments and there are sure to be other new and interesting uses for the language in the future, especially now that embeddable interpreters are available.

In this book, we have attempted to snapshot the browsers that exist currently, which need to be supported by web sites, and collate that information together in a form that has broad scope and is deep enough to be useful on a day-to-day basis. As the language is growing all the time, this is likely to be an ongoing task.

Who is This Book For?

The book is aimed at people who already have some knowledge of JavaScript and need a companion volume to their daily work. It is primarily aimed at the experienced practitioner, and so does not attempt to be a tutorial for the beginner.

> *For a tutorial book, we suggest Paul Wilton's* Beginning JavaScript
> *(Wrox Press, ISBN 1-861004-06-0).*

Typical uses of the book include times when you:

- Need to check out the specific details of a particular language construct or object property
- Know what you want to do, but want to know how JavaScript helps you achieve that
- Want information on cross-browser compatibility issues for your script
- Have encountered a problem in your script and need help to debug it

One important motivation I had for writing this book was to reduce the amount of material I have to carry around when I'm working on projects in my clients' offices. My library now contains several shelves devoted purely to JavaScript, and in researching this book, I ended up with many megabytes of material. There have been many fine books written about JavaScript but I simply cannot carry them around on the train, even with a large rucksack! So, I set out to try and distil enough useful information into one book and organize it so that the information is easy to find. I've also put in material on issues that I've encountered in discussions with other programmers.

The Structure of the Book

To make it easy to navigate through the topics, titles describe the topic content and the topic type and are organized alphabetically. Where a topic might be referred to using several headings, a brief entry in the cross-reference at the end of the book shows the main topic for that subject.

I used a great deal of software automation to manage the book content and the whole thing was built in a database and exported as an XML file set using the DocBook DTD. There are now in excess of 3500 individual topics in this work. That is more than twice as many as we have room for in the printed book, so we've had to put a useful subset of the reference into the printed book, and the complete set of material onto the CD-ROM, which is available both in PDF and HTML formats. Some additional reference information that is not strictly part of the JavaScript language, but that you may find useful, is also included, such as country codes and MIME types.

Where we discuss an object all the important properties, methods, events, and any supporting material are broken out into their own topics, and these detailed entries are included on the CD. Where objects inherit properties and methods, they are listed in the object coverage, but to avoid duplication the information about the inherited properties is described as a member of the super-class. This slightly detracts from the lexical referencing but it saves space. In some cases these inherited properties/methods are deemed important enough to merit a cross-referencing entry of their own.

This allows us to indicate availability of features at a very fine level of detail. Within each topic we can also discuss bugs, gotchas, and areas of difficulty in a focused way.

Language syntax is illustrated by way of example code fragments that show how to access an object, method, or property. More extensive examples are given where necessary.

Because of the scoping rules, properties are available without the need for the `window` object to be specified as a prefix. Thus `navigator` as a topic is available under the `window.navigator` topic as well. Once you have found an entry topic, you can then use the cross-referencing listings to locate other related material.

The book content was developed inside a database system, which provided tools to relate topics. The benefit is a rich source of cross-referencing links between topics. The cross-reference in the printed book is complete; that is, it also includes entries found only on the CD. The italicized cross-references in the printed book can also be found in the printed book.

We will now look at some of the 'features' of JavaScript programming, as an introduction to what topics in the book will address.

Differences between Browsers

For some time, the most popular browsers have been Netscape Navigator 4.7 and Microsoft Internet Explorer 5.0 (MSIE). Other, newer browsers make a point of being standards compliant and so if your script conforms to the standards for core JavaScript as laid down by ECMA and the W3C DOM specifications, it should function correctly.

However, the dominant browsers have for a long time been competing with one another to add new features. Architecturally, this means their browsers have each gone in a completely different direction. The penalty has been that support for various language features has been implemented in each browser in ways that makes it difficult to use in a portable way. Indeed, to make use of some features requires twice the work, since the same code has to be written in two different ways and called after detecting which browser is being used.

Because of the proliferation of browser versions and platforms, features are generally referred to as being available in the revision in which they were first introduced. As the Netscape browser is available on so many different platforms, to test for compliance across all platforms would require a test suite of a dozen machines and 30 or more different installations of browser applications. Indeed, when building such a test suite just prior to starting on this book, I found more than a dozen distinctly different browser versions just for the Macintosh platform and many more than that for Windows. Similarly, MSIE comes in a bewildering variety of versions and platform variants. In addition, the JScript interpreter is a replaceable component that can be upgraded without changing the containing browser.

Browsers and Standards

There is still much that is ambiguous or not yet defined in the standards and the browser manufacturers continue to add new features in competition with one another. Even though they are standards compliant at a functional level, there are still significant differences if you 'look under the hood'.

We have included coverage of the following standards:

- ECMAScript core language up to edition 3 of the standard
- DOM coverage to level 1
- Some DOM coverage of level 2 where implemented in Netscape 6
- Discussion of the features being added at DOM level 3

JavaScript implementations we cover include:

- Netscape 3.0, 4.0, 4.05, 6.0 (the final release came out as we went to press)
- MSIE version 3.0, 4.0, 5.0, 5.5
- Opera 3, 4, 5
- Netscape Enterprise Server

By implication that means we cover JavaScript versions up to 1.5 and JScript up to 5.5. The coverage of Netscape 6.0 is based on it supporting the W3C DOM standards and several bugs in the currently released version prevented the verification of some functionality although that may be platform dependent. There are also some new and unexpected features.

We concentrate our discussions on the peculiarities of Netscape and MSIE because the other browsers that support JavaScript attempt to provide a fault-free standards-based implementation. Since this is a sub-set of the functionality of Netscape and MSIE, other platforms should be adequately covered.

Features and Versions

There are now a wide variety of sources of information about JavaScript and they don't all agree. In particular there is some uncertainty over which release of JavaScript introduces certain features.

The source material was assimilated by examining the standards documents and by inspecting objects with fragments of JavaScript. Then, the availability of features was checked against several alternative reference works. Occasionally, when a consistency error showed up, it was necessary to go back to the browser and test for the availability of a property or method.

Where there is some room for doubt, we have documented the release at which the feature became useful. This is because in earlier releases it may have had a serious flaw or been significantly revised later to make it work properly. Any implementation prior to that may be unreliable. So where we may appear to disagree with other commentators our coverage is based on whether it is practical to use a particular feature at a certain release.

Some browser features are available at an earlier release on some platforms than others are. We take the Windows 32-bit release as our baseline although significant testing was also done on the Macintosh versions, which disappointingly lagged somewhat in performance and feature availability. Both platforms exhibited instabilities and crash-prone behavior but in quite different areas of the language.

As there are so many variants of the browsers, the availability matrix for objects and their member properties/methods is huge and requires a large amount of work to test on all the available combinations. So far, no single reference source has proven to be error free and whilst the information here has been examined and cross-checked it is still likely that there are errors. If, in your work, you disagree with the information provided here, please send feedback (see the end of the *Introduction* for how to do this).

Core JavaScript

At first glance, the JavaScript environment appears to be built around a small core of objects and it is easy to fall into the trap of assuming the language is small and compact. That is certainly true if you are only considering core JavaScript functionality. The core language is defined by ECMA and both Netscape and MSIE both claim to be ECMA compliant. They may well be, but you cannot write much useful JavaScript for deployment in a web page by confining yourself only to the functionality of the ECMA standard. It is at that point that the two browsers begin to diverge.

DOM Support

Likewise, both browsers (MSIE 5 upwards and Netscape 6) claim to be DOM compliant. Browser support for the DOM is slowly converging but if you need to do any esoteric code development that involves DOM traversal and class names, they are still somewhat different.

MSIE implements a DOM model that is structurally right, but the class names of the objects that comprise that model are certainly not correct and do not conform to the DOM standard. Netscape 6 implements a DOM compliant model that does use the correct class names. Another slight difference is that MSIE implements distinctly different classes for some objects whereas Netscape Navigator instantiates the same class for several purposes.

These differences don't cause much grief to you when you are constructing simple scripts and web page enhancements but can be quite a problem if you need to manipulate the DOM structure and operate on objects by means of their class names. This difference did not become apparent until I used inspection scripts to examine internal document structures.

There are also areas where DOM specifies objects in a way that the browsers can implement ambiguously. For example, DOM describes documents as being a generic document class with an HTML document as a sub-class. Browsers simply provide a single document class with no access to the two separate class types.

Object Classes

You might also assume that there is a small and finite set of different object types. However if you inspect the constructor properties and examine the function names, you will find the opposite, there are a large number of object types. For example, the `applets` property that returns a list of applets in a document will give you an `AppletArray` object and not a `Collection` object in Netscape. Trying to work out class names on MSIE is a bit more problematic and it tends to provide generic `Collection` objects instead. By building fragments of JavaScript to inspect objects, you can determine these class names and learn a lot about how the browser maintains the internal model of the page.

The topics are constructed around a browser-centric model. The objects are defined based on their instantiation by an HTML tag in a web browser window. MSIE creates a distinct object class for each tag. Netscape does a similar trick, but not so convincingly in earlier versions. At version 6, the objects are DOM compliant and named differently to those in MSIE and earlier Netscape browsers. Netcsape 6 is so different as to be a new browser with little similarity to the earlier versions of Netscape.

There is an emerging standards-based model that frames the object hierarchy much more logically and, while it is still evolving, it may become a more robust way of describing the catalog of available classes. For now, though, the web and browser dominate use of JavaScript, so this seems like the more appropriate model.

Document Objects

Another area of debate is the `document` object. Typically, the previous documentation describes access to it as if there is only one `document` object. This is true within the context of a single script within a page. However, it is not necessarily true of a window in a web browser. A window may contain many frames or layers. Each one will have its own private `document` object. If you are writing scripts that operate across multiple frames or windows, you may refer to several `document` objects, so the syntax examples are designed to accommodate the different ways in which objects can be accessed.

The Future

JavaScript is becoming available in an ever-wider variety of applications. It is used in:

- PDF forms for validation
- For modifying the behavior of the GUI in developer tools
- Embedded interpreters in cell phones and television set-top boxes

There was not space enough or time to cover these extensively. They are also changing continually and will not be stable enough to document for a while yet.

What Do I Need to Use This Book?

All that is needed to use this book is a text editor and a JavaScript-enabled browser, such as Microsoft Internet Explorer or Netscape Navigator.

To use the CD you will need a browser to read the HTML files and a copy of Adobe Acrobat Reader/Adobe Acrobat eBook Reader to read the PDF files, which are freely available from www.adobe.com. To make navigation easier, the PDF files contain interactive bookmarks, thumbnails, and hyperlinks in the entries.

All of the code examples given in the book are available on the CD, and are also available to download from our web site, www.wrox.com.

Conventions Used in This Book

The convention used for syntax naming is that a variable created within the local scope would be prefixed with `my` while a global variable would be prefixed with `the`. Parameters passed into function and method calls are prefixed with `a`, `an`, or `some`.

The syntax description for an object shows how a reference to an object of that class can be retrieved via a property or method on another object. The syntax for properties and methods show them as members of an object that is referred to with a variable. This manifests itself as an object reference like this:

```
myDocument = document
myDocument = myElement.parentNode
myDocument = myFrame.document
myDocument = myLayer.document
```

Then a property reference looks like this,

```
myDocument.cookie
```

and not:

```
document.cookie
```

Of course you can omit the indirection through a referencing variable and any of these would be equally valid:

```
document.cookie
myElement.parentNode.cookie
myFrame.document.cookie
myLayer.document.cookie
```

But by using the indirection, the syntax descriptions for the member properties and methods are simplified.

In the tables, we have used the abbreviations N for Netscape, NES for Netscape Enterprise Server, and IE for Internet Explorer.

As for styles in the text:

- Filenames, and code in the text appear like so: `dummy.xml`
- Test on user interfaces, and URLs, are shown as: File/Save As…

Customer Support

Wrox has three ways to support books. You can:

- Post and check for errata at www.wrox.com
- Enroll at the peer-to-peer forums at p2p.wrox.com
- Email technical support a query or feedback on our books in general

Errata

You can check for errata for the book at our web site; www.wrox.com, simply navigate to the page for this book. There will be a link to the list of errata.

P2P Lists

You can enroll in our peer to peer discussion forums at p2p.wrox.com. The JavaScript list is available in the 'Web Design' section.

Email Support

If you wish to point out an errata to put up on the website or directly query a problem in the book with an expert who knows the book in detail, then e-mail `support@wrox.com`. A typical email should include the following things:

- The name of the book, the last four digits of the ISBN and the entry name for the problem in the Subject field
- Your name, contact info and the problem in the body of the message

You may want to tell us your opinion of this book, or you may have ideas about how it can be improved, in which case, e-mail `feedback@wrox.com`. We will do our utmost to act upon your comments.

A object (Object/HTML)

An object that represents an <A> element when instantiated in MSIE.

Availability:	JScript –1.0 Internet Explorer –3.02	
Inherits from:	Element object, Node object	
JavaScript syntax:	IE	*myA* = *myDocument*.all.*anElementID*
	IE	*myA* = *myDocument*.all.tags("A")[*anIndex*]
	IE	*myA* = *myDocument*.all[*aName*]
	IE	*myA* = *myDocument*.anchors.item(*aName*)[*anIndex*]
	-	*myA* = *myDocument*.anchors[*aName*]
	-	*myA* = *myDocument*.anchors[*anIndex*]
	-	*myA* = *myDocument*.getElementById(*anElementID*)
	-	*myA* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myA* = *myDocument*.getElementsByTagName("A")[*anIndex*]
	IE	*myA* = *myDocument*.links.item(*aName*)[*anIndex*]
	-	*myA* = *myDocument*.links[*aName*]
	-	*myA* = *myDocument*.links[*anIndex*]
HTML syntax:	<A> ... </A>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	accessKey, dataFld, dataSrc, hash, host, hostname, href, Methods, mimeType, nameProp, pathname, port, protocol, protocolLong, rel, search, tabIndex, target	
Event handlers:	onBlur, onClick, onDblClick, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

See also:	Element object, Input.accessKey, Map object, Anchor object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	-	3.0 +	-	4.0 +	-	1 +	-	-
dataFld	-	1.0 +	-	3.02 +	-	-	-	-
dataSrc	-	1.0 +	-	3.02 +	-	-	-	-
hash	-	1.0 +	-	3.02 +	-	-	-	-
host	-	1.0 +	-	3.02 +	-	-	-	-
hostname	-	1.0 +	-	3.02 +	-	-	-	-
href	-	1.0 +	-	3.02 +	-	1 +	-	-
Methods	-	1.0 +	-	3.02 +	-	-	-	-
mimeType	-	1.0 +	-	3.02 +	-	-	-	-
nameProp	-	1.0 +	-	3.02 +	-	-	-	-
pathname	-	1.0 +	-	3.02 +	-	-	-	-
port	-	1.0 +	-	3.02 +	-	-	-	-
protocol	-	1.0 +	-	3.02 +	-	-	-	-
protocolLong	-	1.0 +	-	3.02 +	-	-	-	-
rel	-	1.0 +	-	3.02 +	-	-	-	-
search	-	1.0 +	-	3.02 +	-	-	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
target	-	1.0 +	-	3.02 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onBlur	-	3.0 +	-	4.0 +	-	-	-	Warning
onClick	-	1.0 +	-	3.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onFocus	-	3.0 +	-	4.0 +	-	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	1.0 +	-	3.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

ABBR object (Object/HTML)

An object representing the HTML content (an abbreviation) delimited by the <ABBR> HTML tags.

Availability:	HTML version – 4.0 JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object, `Node` object	
JavaScript syntax:	IE	*myABBR* = *myDocument*.all.*anElementID*
	IE	*myABBR* = *myDocument*.all.tags("ABBR")[*anIndex*]
	IE	*myABBR* = *myDocument*.all[*aName*]
	-	*myABBR* = *myDocument*.getElementById(*anElementID*)
	-	*myABBR* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myABBR* = *myDocument*.getElementsByTagName("ABBR")[*anIndex*]
HTML syntax:	<ABBR> ... </ABBR>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Event handlers:	onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp	

See also:	`style.speak`, `Element` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

about: URL (Request method)

This is a special kind of URL that fetches content from a storage area inside the Netscape browser instead from using HTTP to get it from a web server.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0

This is a special request method provided by the Netscape browser to gain access to local client-side resources. The resources are loaded from inside the application itself.

In the Macintosh version of Navigator, this means they are stored in the resource fork of the browser application. If you need to deploy a custom version of Navigator within an intranet environment, with some care you can modify these resources with a resource-editing tool, such as ResEdit. Always work on a copy of the application and test the changes thoroughly.

On other platforms, the resources are likely to be stored in files located in folders adjacent to the application. You will need to study your own copy of Netscape to see what you can change.

These special URLs are mostly not present in early versions of MSIE, although there will be some internal resources, which may provide customization opportunities. MSIE also supports an `about:blank` URL that provides a blank page. There may be others hidden away inside the application.

You may also be able to obtain administration tools from Netscape and Microsoft to carry out legitimate customizations on the browsers before deploying them throughout your organization.

The following special URLs seem to work when typed into the location box:

URL	Description
`about:logo`	Netscape logo
`about:mozilla`	A fire & brimstone quote from the book of Mozilla (Yes it's really there – at least on some versions)
`about:authors`	Shows a cryptic message about the page having been removed, although the `authors.html` file is still present inside the application
`about:cache`	Displays a disk cache report
`about:document`	Displays the document info console
`about:fonts`	Displays the font info console
`about:global`	A global history report
`about:image-cache`	A report on the internal image cache
`about:license`	A hyperlink to the Netscape license document
`about:mailintro`	Displays the Netscape mail info page

URL	Description
about:memory-cache	A report on the memory cache
about:pics	Generates a security exception
about:plugins	A page of information about the plugins
about:security?advisor=XXX	Brings up a security console where XXX indicates the window to operate on.
about:security?banner-insecure	Serves an unlocked padlock image
about:security?banner-secure	Serves a locked padlock image
about:security?issuer-logo=XXX	Returns a graphic where XXX identifies which one
about:security?subject-logo=XXX	Returns a graphic where XXX identifies which one
about:coslogo2	Cosmo logo
about:fclogo	Full Circle software logo
about:hslogo	Beatnik logo
about:hype	An audio clip
about:insologo	Inso logo
about:javalogo	Java compatible logo
about:litronic	Litronic logo
about:mclogo	Marimba Castanet logo
about:mmlogo	Macromedia logo
about:ncclogo	Netcast logo
about:odilogo	Object Design logo
about:qtlogo	Apple QuickTime logo
about:rsalogo	RSA secure logo
about:symlogo	Symantec logo
about:tdlogo	TrueDoc logo
about:visilogo	VisiGenic logo
about:blank	Presents a blank page on Netscape Navigator 3 and MSIE version 5; used to create a blank page when a new window is opened

Some of these URLs can be used in frames, but others can't. A few can be used as HREF values. JavaScript complains that the `about: request` method is illegal. This means you cannot change the `location.href` within a page to any of the "`about:`" URLs. However, you might be able to write some `innerHTML` content into a `<DIV>` or `<SPAN>` to place a link to these assets.

Many of the built-in assets are used as image sources in the `about` page. It's possible you might want to display the Netscape logo. If you are aware that you are using software provided by the other third parties, you might (if they give you permission) place their logo on the screen when you are using features of their software. You should ask first, although Netscape probably won't mind their logo being served like this.

The interesting thing about this is that you are effectively serving assets out of a static cache in the client file system.

The URL that points at the license document may be useful as it is possible you might want to display the Netscape license if you are redistributing the browser.

The `about:plugins` URL yields a page containing some useful JavaScript that displays the plugins page. You may find some useful techniques in here for managing plugin facilities although they may be Netscape compatible only.

Mostly, these special URLs will be useful for debugging. Getting details of the disk cache, for example, may be useful. Pulling up the JavaScript debugger page if you detect an error in your script might also be a cool trick.

The MSIE and Netscape browsers can both use the `about:blank` URL value as a default page when the browser is started up.

Warnings:

- The `UniversalBrowserRead` privilege is required for access to internal browser values and state information such as the cache contents.

See also:	`javascript:` URL, `nethelp:` URL, `UniversalBrowserAccess`, `UniversalBrowserRead`, URL

abstract (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

AbstractView object (Object/DOM)

An object that belongs to the DOM level 2 views module.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myAbstractView* = *myMouseEvent*.view
	N	*myAbstractView* = *myUIEvent.view*

This is part of a new suite of functionality introduced at DOM level 2, which provides a way of looking at documents from alternative points of view. At present only the Abstract and Document views are standardized and, because the capabilities are quite new, implementations may be incomplete at this stage.

See also:	`MouseEvent` object, `MouseEvent.initMouseEvent()`, `UIEvent` object

Accessor method (Definition)

A method for accessing publicly available object properties.

Availability:	ECMAScript edition – 2

A method used to store or retrieve property values contained in objects.

In ECMAScript-compliant implementations, this is accomplished with internal functions named `Get()` and `Put()`.

If you add new properties to an object of your own, you may want to implement functions that operate by using the '`this`' variable to access properties. These functions are then associated with the object or its prototype, so that they can be shared. They are then referred to as methods rather than functions.

See also:	`function( ... )` ..., `Get()`, Method, `Put()`

Cross-references:

ECMA 262 edition 2 – section – 8.6.2

ECMA 262 edition 3 – section – 8.6.2

ACRONYM object (Object/HTML)

An object representing the HTML content delimited by the `<ACRONYM>` HTML tags.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object, `Node` object	
JavaScript syntax:	IE	*myACRONYM* = *myDocument*.all.*anElementID*
	IE	*myACRONYM* = *myDocument*.all.tags("ACRONYM")[*anIndex*]
	IE	*myACRONYM* = *myDocument*.all[*aName*]
	-	*myACRONYM* = *myDocument*.getElementById(*anElementID*)
	-	*myACRONYM* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myACRONYM* = *myDocument*.getElementsByTagName("ACRONYM")[*anIndex*]

HTML syntax:	`<ACRONYM> ... </ACRONYM>`	
Argument list:	*anElementID*	The ID value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

See also:	`style.speak`, `Element` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Activation object (Object/internal)

The `activation` object is created when the flow of control first enters an execution context.

Availability:	ECMAScript edition – 2

The `activation` object is created when the flow of control first enters an execution context for declared function code, anonymous code or implementation-supplied code.

As the `activation` object is created, it is associated with the execution context. On initialization, it has a property called `arguments` that cannot be deleted and that refers to an `arguments` object.

This `activation` object is then used as the variable object for instantiating all the argument variables. The `activation` object is discarded when the function returns its result to the caller. The `activation` object is an internal mechanism and so cannot be passed to the outside world, although members of the `activation` object may well be accessible to a running script.

See also:	Execution context, `function(…)` …

Cross-references:

ECMA 262 edition 2 – section – 10.1.6

ECMA 262 edition 3 – section – 10.1.6

Active Server Pages (Product)

A Microsoft web server product.

This is a server-side programming framework that supports JavaScript. More accurately, it supports JScript, which is Microsoft's flavor of JavaScript.

For more information, see *ASP 3.0 Programmer's Reference*, ISBN 1-861003-23-4 from Wrox Press.

See also:	`BODY.recordNumber`, `Event.bookmarks[]`, `Event.boundElements[]`, `Input.recordNumber`, `SCRIPT.recordNumber`, Server-side JavaScript, `TextStream` object

ActiveX (Product)

This is a Microsoft technology for embedding and sharing code.

In MSIE (on Windows), interactions between scripts and applets takes place by means of ActiveX. Microsoft prefers to treat applets as a special kind of `ActiveX` object. Netscape shows a similar preference towards treating applets as Java components.

If you are developing web-based applications for a captive audience who you know will be running MSIE on Windows, then this technology may be appropriate for your project.

However, ActiveX is not supported on Netscape and, in fact, is unlikely to be well supported on any other browsers aside from MSIE.

Coupling this with the fact that it is not supported outside the Windows platform, you will almost certainly find Java to be a more portable solution. The Java solution is also secured better than ActiveX, which can expose the internals of your system in ways you would rather avoid.

For a high degree of Windows integration and a very Microsoft-oriented solution, ActiveX is ideal. For portability across platforms and browsers, it's likely you'll do much better to select Java.

See also:	`Applet` object, `Dictionary` object, Glue code, LiveConnect

Web-references:

http://msdn.microsoft.com/scripting/

ActiveXObject object (Object/JScript)

A Windows and MSIE specific object that allows various document components to be embedded.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`myActiveX = ActiveXObject`
	IE	`myActiveX = new ActiveXObject (anApplication)`
Argument list:	*anApplication*	References an external application
Collections:	Depends on the object created by the constructor	

This is an object for embedding other applications into web pages on the Windows platform. The example shows the creation of an object that is managed by the Word application.

This is also used to create `Dictionary` objects by using the Scripting application to create a new `Dictionary` object.

Warnings:

- This is totally non-portable and non-standard, but if your scripts are likely to be deployed in a Windows-only environment, it may be useful.
- Using this construct in client-side scripting is subject to security restrictions. If a script in a web browser could just instantiate Word, then that implies that it has rights of access to the local file system. The normal IE security settings disallow that level of access.

Example code:

```
// An example that opens a Word document and writes
// text into it.
var myActiveX = new ActiveXObject("Word.Document");
myActiveX.Application.Visible=true;
myString="Some text to be written to the document";
// now write the text to the word document
myActiveX.application.selection.typeText(myString);
```

See also:	`Dictionary` object, `OBJECT` object

ActiveXObject() (Constructor)

Used for manufacturing new `ActiveX` objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`new ActiveXObject(`*`anObjectType`*`)`
	IE	`new ActiveXObject(`*`anObjectType`*`, `*`aLocation`*`)`
Argument list:	*`anObjectType`*	What sort of application and object class type to be created
	`aLocation`	A server name where the source object is located

You can use this constructor for creating new objects. You need to specify the kind of object to be created in the string argument value.For example, to create a Microsoft Word document, pass the string "Word.Document" to the constructor.

You can also specify a second optional argument to locate the application on a remote server.

Here are some example applications you can invoke:

- Word.Document – Create an empty Word document
- Excel.Sheet – Create an empty Excel spreadsheet
- Microsoft.XMLDOM – Create a new XML document

Other alternatives depend on the applications you have installed on your client system.

See also:	`GetObject()`

Add (+) (Operator/additive)

Add two numeric operands together. See concatenation for Strings.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`anOperand1 + anOperand2`*
Argument list:	*`anOperand1`*	An expression that evaluates to a number
	`anOperand2`	Another expression that evaluates to a numeric value

The addition operator adds two numeric values together or concatenates one string onto another.

When used with numeric operands, the plus sign adds the values together.

The addition is commutative, meaning that the order of the operands does not affect the outcome of the calculation. However, the calculation is not always associative (so `(a+b)+c` is not always the same as `a+(b+c)`) and so the precedence established with the grouping operator might affect the outcome.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

If either operand is `NaN`, the result will be `NaN`.

The sum of infinity and minus infinity will be `NaN`; they do not cancel one another out.

The sum of two infinity values of the same sign will be the infinity of that sign.

The sum of infinity and a finite value is equal to the infinite operand.

Internally the sum of two negative zero values is -0. However, the sum of two positive zero value or a positive and negative zero value added together will be +0. At the scripting level however, you cannot determine whether a zero is positive or negative, but its sense may affect subsequent computations.

The sum of zero and a non-zero value will be the non-zero value.

The sum of two non-zero finite values of the same magnitude but opposite signs will be zero.

Provided neither an infinity, a zero nor `NaN` is involved, adding two finite values results in the sum of the two values given that the result will be rounded to its nearest representable value. Where the result exceeds the largest presentable value, infinity will be substituted. A negative infinity may result from an underflow.

The addition/concatenation operator looks at the arguments and if either is a String already or preferentially converts to one, then a concatenation occurs. If neither operator prefers to be a String, then a Number conversion happens and the values are added.

See also:	Add then assign (+=), Additive expression, Additive operator, Associativity, Negation operator (-), Operator Precedence, String concatenate (+), Subtract (-), Type conversion, Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.6.1

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.6.1

Wrox *Instant JavaScript*, ISBN 1-861001-27-4 – page – 37

Add then assign (+=) (Operator/assignment)

Add two numeric operands and assign the result to the first. See concatenation for Strings.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`anOperand1 += anOperand2`*
Argument list:	*`anOperand1`*	An expression that evaluates to a number
	`anOperand2`	Another numeric value

Add the right operand to the left operand and assign the result to the left operand.

This is functionally equivalent to the expression:

`anOperand1` = *`anOperand1`* + *`anOperand2`*;

Although this is classified as an assignment operator, it is really a compound of an assignment and an additive operator.

It also works with string values and will concatenate the second onto the first.

The associativity is right to left.

Refer to the operator precedence topic for details of execution order.

The new value of *`anOperand1`* is returned as a result of the expression.

Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

Example code:

```
// Initialize with numeric valuesmyVar1 = 100;myVar2 = 1000;
// After this myVar1 contains 1100, myVar2 is unchanged
myVar1 += myVar2;
```

See also:	Add (+), Additive operator, Assign value (=), Assignment expression, Assignment operator, Associativity, Concatenate then assign (+=), Increment value (++), LValue, Operator Precedence, Subtract then assign (-=)

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

Adding JavaScript to HTML (Advice)

The most popular use of JavaScript is in web pages. Adding it is quite simple.

To add JavaScript to a web page, you can use the following techniques:

- The `<SCRIPT>` HTML tag containing the script source text
- From an external file using the `<SCRIPT SRC="filename.js">` HTML tag and attribute
- From an external file using the `<SCRIPT ARCHIVE="" SRC="">` HTML tag and attributes
- With an event handler attribute
- In a `javascript:` URL
- By means of a JavaScript style sheet
- As a JavaScript entity value for an HTML attribute
- Within a conditional comment

Refer to the individual topics for specific details covering each case.

See also:	`<SCRIPT ARCHIVE="...">`, `<SCRIPT SRC="...">`, `<SCRIPT>`, `<STYLE TYPE="...">`, Conditional comment, Event handler, JavaScript entity, `javascript:` URL

Additive expression (Definition)

This is an expression that adds or subtracts values.

Availability:	ECMAScript edition – 2

Additive expressions use the additive operators to yield a result by operating on two values, which may themselves be expressions.

See also:	Add (+), Decrement value (--), Expression, Increment value (++), Negation operator (-)

Cross-references:

ECMA 262 edition 2 – section – 11.6

ECMA 262 edition 3 – section – 11.6

Additive operator (Definition)

An operator that adds or subtracts values.

Availability:	ECMAScript edition – 2

Here is a table summarizing all operators that can be classified as additive, even those which are primarily classified in other categories:

Value	Meaning
`+`	Add
`-`	Subtract
`+=`	Add and assign
`-=`	Subtract and assign
`++`	Increment
`--`	Decrement

Additive operators perform numeric addition and subtraction or string concatenation depending on the native type of the operands.

It might seem perverse to call a subtraction symbol an additive operator, but the word additive is used in the same context as multiplicative when talking about division. That is, a negative value is added to perform subtraction. It's all about the kind of logic used in the interpreter kernel.

See also:	Add (+), Add then assign (+=), Arithmetic operator, Associativity, Decrement value (--), Increment value (++), Negation operator (-), Operator, Operator Precedence, Positive value (+), Postfix expression, Prefix decrement (--), Prefix expression, Prefix increment (++), String concatenate (+), String operator, Subtract (-), Subtract then assign (-=)

Cross-references:

ECMA 262 edition 2 – section – 11.6

ECMA 262 edition 3 – section – 11.6

ADDRESS object (Object/HTML)

An object representing the HTML content delimited by the `<ADDRESS>` HTML tags.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Inherits from:	`Element` object, `Node` object

Table continued on following page

JavaScript syntax:	IE	*myADDRESS* = *myDocument*.all.*anElementID*
	IE	*myADDRESS* = *myDocument*.all.tags ("ADDRESS")[*anIndex*]
	IE	*myADDRESS* = *myDocument*.all[*aName*]
	-	*myADDRESS* = *myDocument*.getElementById(*anElementID*)
	-	*myADDRESS* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myADDRESS* = *myDocument*.getElementsByTagName ("ADDRESS")[*anIndex*]
HTML syntax:	<ADDRESS> ... </ADDRESS>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an *Element* object
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

See Also	Element object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

ADO (Product)

ActiveX Data Objects is a Microsoft technology for accessing data.

This is a technology that enables an ASP server to access data from a data source. It provides an easy-to-use object interface to the OLE database access mechanisms.

The ADO object model is built around a set of objects of the following kinds:

- `Command`
- `Connection`
- `Record`
- `Recordset`
- `Stream`

The ADO object model includes the following collections containing objects of these types:

- `Errors`
- `Fields`
- `Parameters`
- `Properties`

This is all covered in greater depth in the Wrox book *ASP 3.0 Programmers Reference*, ISBN 1-861003-23-4.

See also: `BODY.recordNumber`, `Event.bookmarks[]`, `Input.recordNumber`, `SCRIPT.recordNumber`, Window furniture

Adornments (Definition)

The various control items that form the window border and can be selectively enabled as needed.

Aggregate type (Definition)

Data types built from several atomic components.

An aggregate data type is built by combining one or more atomic data types to build a more sophisticated data type. In compiled non-object-oriented languages one might create structures as aggregates of member variables. These are analogous to object classes.

Arrays are another example of an aggregate data type.

Other aggregate types include the various collection-based objects.

See also: `Array` object, `Cast` operator, `Function` object, `Object` object, Scalar type, Type

alert() (Method)

Present the alert dialog box to the user.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`alert(`*`aString`*`)`
	-	*`myWindow`*`.alert(`*`aString`*`)`
Argument list:	*`aString`*	Some text to display in the alert box

See Also	`Window.alert()`

Alias (Definition)

An indirect reference to an object.

By assigning an object to a variable, you are not copying that object but instead making a reference to it. A reference is sometimes called an alias. The same technique is used in Java, and in non-object-oriented languages you accomplish something similar with pointers.

See also:	Object

Alpha() (Filter/visual)

A visual filter for controlling transparency.

Availability:	JScript – 3.0 Internet Explorer – 4.0

AlphaImageLoader() (Filter/visual)

An image is displayed in the object, with some additional control over how it is displayed.

Availability:	JScript – 5.5 Internet Explorer – 5.5
See Also	Filter – `AlphaImageLoader()`

Anchor object (Object/HTML)

An object representing an HTML `<A>` tag.

Availability:	DOM level – 1 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Inherits from:	`Element` object, `Node` object	
JavaScript syntax:	-	*myAnchor* = *myAnchorArray*[*aName*]
	-	*myAnchor* = *myAnchorArray*[*anIndex*]
	-	*myAnchor* = *myDocument*.anchors[*aName*]
	-	*myAnchor* = *myDocument*.anchors[*anIndex*]
	-	*myAnchor* = *myDocument*.getElementById(*anElementID*)
	-	*myAnchor* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myAnchor* = *myDocument*.getElementsByTagName("A")[*anIndex*]
	-	*myAnchor* = *myDocument*.links[*aName*]
	-	*myAnchor* = *myDocument*.links[*anIndex*]
	IE	*myAnchor* = *myDocument*.all.*anElementID*
	IE	*myAnchor* = *myDocument*.all.tags("A")[*anIndex*]
	IE	*myAnchor* = *myDocument*.all[*aName*]
	IE	*myAnchor* = *myDocument*.anchors.item(*aName*)[*anIndex*]
	IE	*myAnchor* = *myDocument*.links.item(*aName*)[*anIndex*]
HTML syntax:	`<A> ... </A>`	
Argument list:	*aName*	An associative array reference to the `anchor` object.
	aName	The `name` property of the `anchor` object
	anIndex	An index into the `anchors` collection
	someText	The text (or `innerText`) property of the `anchor`
	anElementID	The ID value of an `Element` object

Table continued on following page

Object properties:	`accessKey`, `charset`, `coords`, `dataFld`, `dataSrc`, `hash`, `host`, `hostname`, `href`, `hreflang`, `Methods`, `mimeType`, `name`, `nameProp`, `pathname`, `port`, `protocol`, `protocolLong`, `recordNumber`, `rel`, `rev`, `search`, `shape`, `tabIndex`, `target`, `text`, `type`, `urn`, `x`, `y`
Object methods:	`blur()`, `focus()`
Event handlers:	`onClick`, `onMouseDown`, `onMouseOut`, `onMouseOver`, `onMouseUp`

See Also:	Filter – `Alpha()`

This object represents a named location in the HTML document. Only those `<A>` HTML tags that have a `NAME` attribute will have `anchor` objects created for them. All the anchors are listed in the `anchors[]` array object that belongs to the `document` object that represents the HTML.

Although the `<A>` tag is also used to create links using the `HREF` attribute, they are not anchors unless they are named. Any `<A>` tags that have `HREF` attributes (whether or not they have `NAME` attributes) will be listed in the `links[]` array.

In Netscape, you can construct new instances of the `Anchor` object, but there is no constructor property in MSIE to support this.

`<A>` tags and the objects that represent them are inline elements. Placing them into a document does not create a line break.

Warnings:

- If you put an `anchor` object into a `document.write()`, in Netscape you get a string containing the object class. In MSIE, you will get the `HREF` string if there is one and an empty string if there isn't.
- MSIE provides access to properties that would normally be considered part of the `link` object. Internally MSIE probably maintains a single object type for anchors and links, whereas Netscape implements two quite different classes.
- Netscape supports an associative reference to an `anchor` object within the `anchors[]` array according to the value of its `NAME` tag attribute. MSIE does not support this means of locating an `anchor` object in quite the same way.
- Note that although the syntax examples illustrate the use of an `innerText` property, Netscape does not support this mode of access and it will generate an error.

Example code:

```
<!-- Example showing how to dynamically replace -->
<!-- the anchor text -->
<HTML>
<HEAD></HEAD>
<BODY>
<A NAME="A1" HREF="www.apple.com">Click here</A>
<BR>
```

```
<A NAME="A2" HREF="www.wrox.com">Click here</A>
<BR>
<A NAME="A3" HREF="www.msdn.com">Click here</A>
<BR>
<BR>
<HR>
<SCRIPT>
myLength = document.anchors.length;
for (myEnumerator=0; myEnumerator<myLength; myEnumerator++ )
{
   document.anchors[myEnumerator].innerText =
document.anchors[myEnumerator].name;}
</SCRIPT>
</BODY>
</HTML>
```

See also: `Document.anchors[]`, `Document.links[]`, `Element` object, `Element.all[]`, `Input.accessKey`, `LINK` object, `Location` object, `String.anchor()`, Subclasses, Superclasses, URL, `Url` object, `Window.scrollTo()`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
charset	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
coords	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
dataFld	-	3.0 +	-	4.0 +	-	-	-	-
dataSrc	-	3.0 +	-	4.0 +	-	-	-	-
hash	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
host	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
hostname	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
href	1.2 +	3.0 +	4.0 +	4.0 +	-	1 +	-	Warning
hreflang	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
Methods	-	3.0 +	-	4.0 +	-	-	-	-
mimeType	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
name	1.2 +	3.0 +	4.0 +	4.0 +	-	1 +	-	Warning
nameProp	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
pathname	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
port	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
protocol	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
protocolLong	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
recordNumber	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
rel	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
rev	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
search	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
shape	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
target	1.2 +	3.0 +	4.0 +	4.0 +	-	1 +	-	Warning
text	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly
type	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
urn	-	3.0 +	-	4.0 +	-	-	-	-
x	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly
y	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
blur()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
focus()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.2+	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOut	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Anchor() (Constructor)

You can construct new `Anchor` objects in Netscape.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Anchor` object	
JavaScript syntax:	N	`new Anchor`
	N	`new Anchor()`
	N	`new` *`myAnchor`*`.constructor`

Although you can construct new anchors in Netscape, inserting them into a document is somewhat problematic since the correct way to do that would be to rewrite a fragment of HTML. You may find that replacing an `anchor` object in the `document.anchors` array links your new object to the `<A>` HTML tag at the appropriate location in the document.

Warnings:

- This is only available on Netscape and is therefore not recommended for use in deployable applications. MSIE generates a run-time error if you attempt to make a new `Anchor` object.

See also:	`Anchor` object, `Anchor.host`

Anchor.accessKey (Property)

A key that needs to be pressed before the `anchor` object will respond to data entry.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myInputObject`*`.accessKey`

This is an extension that allows the anchor elements to be deadlocked unless a certain key is held down.

On some browsers and operating systems, you may need to hold down one of the modifier keys for this to work. The modifier key required depends on the environment you are using.

Warnings:

- This is not supported in some versions of the MSIE browser on Macintosh.

Anchor.blur() (Method)

Remove input focus from the `Anchor` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*`myInputObject`*`.blur()`

This will trigger the `onblur` event handler function attached to the `onblur` property of the object.

Anchor.charset (Property)

This property indicates the character encoding of the document at the location specified by the URL.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.charset`

This would contain the character set being used by the document. For example the value "`iso-8859-1`" is likely to be returned, but the local variant of the browser and OS may affect the value you get.

This property might contain a value such as:

```
csISO5427Cyrillic
```

Details of other aliases can be located at the IANA registry. In that registry are listed the names and aliases of a wide variety of character sets. Even though there are nearly 800 names and aliases, it seems on inspection that there are items missing.

See also:	`LINK.charset,Url.charset`

Web-references:

ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

Anchor.coords (Property)

This defines an area map within an image that is inside the `<AREA>` HTML tags.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.coords`

When a shaped area is defined within an image map, the rectangle around the shape is defined with the `coords` property. The value is defined with the `COORDS` HTML tag attribute.

See also:	`Anchor.shape,Area.coords,Url.coords`

Anchor.dataFld (Property)

This binds the anchor object to a remote data source in MSIE.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myInputObject`*`.dataFld`

This is part of the MSIE data-binding mechanism that associates a column name in the data source with the `value` property of an `Anchor` object. You must also set the `dataSrc` property for the object. Normally, both the `dataFld` and `dataSrc` values would be defined with the `DATAFLD` and `DATASRC` HTML tag attributes in the document source.

Note that the value is case-sensitive and must refer to a column that exists within the data source.

Setting both the `dataFld` and `dataSrc` properties to an empty string will disconnect the element from the database.

Anchor.dataSrc (Property)

The name of a remote ODBC data source is stored in this property.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myInputObject`*`.dataSrc`

This is part of the MSIE data-binding support. It contains the name of an ODBC data source (which might be any kind of SQL database that supports such an adapter). The data source and element are bound together with each column of the data source providing a source value to different element objects through their `dataFld` property.

Normally, both the `dataFld` and `dataSrc` values would be defined with the `DATAFLD` and `DATASRC` HTML tag attributes in the document source.

Setting both the `dataFld` and `dataSrc` properties to an empty string will disconnect the element from the database.

Anchor.focus() (Method)

Brings input focus back to the `anchor` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myInputObject*`.focus()`

The receiving `Anchor` object will receive a Focus event trigger and execute its function referred to by the `onfocus` event handler property.

The element that previously had focus (if any element did) will receive a Blur event trigger.

Anchor.hash (Property)

On MSIE the `Url.hash` property is also available as the `Anchor.hash` property.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.hash`
	-	*myAnchor*`.hash = `*newValue*
HTML syntax:	`<A HREF="...">`	

This yields the hash suffix of the `HREF` value in an `<A>` tag.

You can assign a new value to this property, which will become a new anchor location within the document.

Warnings:

- This attribute may not work correctly when URLs are accessed from one frame to another in some versions of MSIE. You should check your target platforms for compliance.
- If you assign a value to this property in MSIE, you should omit the leading hash.
- Since the `hash` property of an `Anchor` object is not portable in all older browser versions, you should use the `pathname` property of the corresponding `Url` object to be able to work across MSIE and Netscape.
- As features become deprecated, it may become necessary to support both techniques in browser-specific code according to your needs regarding the support of legacy browsers.

See also:	`Anchor` object, `Anchor.host`, `Anchor.hostname`, `Anchor.href`, `Anchor.pathname`, `Anchor.port`, `Anchor.protocol`, `Anchor.search`, `Anchor.target`, URL, `Url.hash`, `Url.host`, `Url.hostname`, `Url.href`, `Url.pathname`, `Url.port`, `Url.protocol`, `Url.search`, `Url.target`

Anchor.host (Property)

On MSIE the `link.host` property is also available as the `anchor.host` property.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myAnchor*.host - *myAnchor*.host = *newHostPort*
HTML syntax:	`<A HREF="...">`

This yields the host and port value of the `HREF` value in an `<A>` tag.

You can redefine the host to request the URL by assigning a new value to this property.

Warnings:

- Since the `host` property of an `Anchor` object is not portable in all older browser versions, you should use the `pathname` property of the corresponding `Url` object to be able to work across MSIE and Netscape.
- As features become deprecated, it may become necessary to support both techniques in browser-specific code according to your needs regarding the support of legacy browsers.

See also:	`Anchor` object, `Anchor()`, `Anchor.hash`, `Anchor.href`, `Anchor.pathname`, `Anchor.port`, `Anchor.protocol`, `Anchor.search`, `Anchor.target`, URL, `Url.host`

Anchor.hostname (Property)

On MSIE the `link.hostname` property is also available as the `anchor.hostname` property.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.hostname`
	-	*myAnchor*`.hostname = `*newHostname*
HTML syntax:	`<A HREF="...">`	

This yields the host value of the `HREF` value in an `<A>` tag.

You can redefine the hostname to request the URL by assigning a new value to this property.

Warnings:

- Be careful not to assign a port number with the host name, otherwise your new URL may acquire two port numbers, which makes it invalid.
- Since the `hostname` property of an `Anchor` object is not portable in all older browser versions, you should use the `pathname` property of the corresponding `Url` object to be able to work across MSIE and Netscape.
- As features become deprecated, it may become necessary to support both techniques in browser-specific code according to your needs regarding the support of legacy browsers.

See also:	`Anchor` object, `Anchor.hash`, `Anchor.href`, `Anchor.pathname`, `Anchor.port`, `Anchor.protocol`, `Anchor.search`, `Anchor.target`, URL, `Url.hostname`

Anchor.href (Property)

On MSIE the `link.href` property is also available as the `anchor.href` property.

Availability:	DOM level – 1 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.href`
	-	*myAnchor*`.href = `*newHref*
HTML syntax:	`<A HREF="...">`	

This yields the entire value of the HREF attribute in an <A> tag.

You can redefine the entire HREF content by assigning a new value to this property.

Warnings:

- Since the href property of an Anchor object is not portable in all older browser versions, you should use the pathname property of the corresponding Url object to be able to work across MSIE and Netscape.
- As features become deprecated, it may become necessary to support both techniques in browser-specific code according to your needs regarding the support of legacy browsers.

See also:	Anchor object, Anchor.hash, Anchor.host, Anchor.hostname, Anchor.pathname, Anchor.port, Anchor.protocol, Anchor.search, Anchor.target, Location.href, URL, Url.href

Anchor.hreflang (Property)

The language code of the document at the location specified by the URL.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myAnchor*.hreflang

This property should contain values that use the international language two-letter abbreviation codes. These are not the same as the country codes, which are also two letter values.

Refer to the Language codes topic for a list of the available language codes.

See also:	Language codes, LINK.hreflang

Anchor.Methods (Property)

A property that can indicate some keywords regarding the action that the server provides when the link is clicked on. These reflect the request methods.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myAnchor*.Methods

The possible values of this property are those of the valid methods for the HTTP protocol. It could be one of the following:

- `GET`
- `HEAD`
- `POST`
- `PUT`
- `DELETE`
- `OPTIONS`
- `TRACE`

It is likely that only the `GET` and `POST` methods make any logical sense in this context. On rare occasions, the `PUT` method may be referred to, although it is unusual to find a web server that accepts documents with this request method. Likewise `DELETE` is normally only supported within very strict constraints.

The method name can be specified in upper or lower case.

See also:	`Url.Methods`

Anchor.mimeType (Property)

Contains a long form human readable version of the MIME type of the document at the location specified by the anchor's URL.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myAnchor`*`.mimeType`

The MSIE browser maps the file extension of the file belonging to the anchor to an extended description of the file format, which it makes available through the `mimeType` property. Here is a list of some `mimeType` values it pays special attention to.

File type:	MSIE expanded Mime type:
`.css`	Microsoft CSS1 Style Sheet (W3C would have been more appropriate)
`.gif`	GIF Image
`.htc`	Microsoft HTML Component file for behaviors
`.htm`	Microsoft HTML Document 4.0
`.html`	Microsoft HTML Document 4.0
`.jpg`	JPEG Image

File type:	MSIE expanded Mime type:
`.js`	Microsoft JScript File
`.txt`	Text Document
`.vbs`	Microsoft VBScript File
`.xxx`	All unrecognized file types are returned as `xxx File` with no further expansion

Microsoft asserts that `.htm` and `.html` files are "Microsoft HTML" and `.css` files are "Microsoft CSS1" style sheets. It also asserts that `.js` files are "Microsoft JScript" files. Microsoft doesn't really own those file extensions across all platforms, nor indeed does it even own them on the Windows platform.

Warnings:

- Do not confuse this value with other `mimeType` properties. For example the `Navigator` object has a `mimeTypes[]` collection property with references to `mimeType` objects. The `mimeType` property of an MSIE `Anchor` object is a simple string primitive value and not a `mimeType` object.

See also:	MIME types, `Url.mimeType`

Property attributes:

`ReadOnly`.

Anchor.name (Property)

This corresponds to the `NAME` attribute of the `<A>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.name`
	-	*myAnchor*`.name = `*aName*
HTML syntax:	`<A NAME="aName">`	
Argument list:	*aName*	A new name for the anchor.

The value of this property is defined by the `NAME` tag attribute in the HTML that describes the document. Without the `NAME` attribute, the `anchor` object does not get added to the `anchors[]` array.

This `name` property contains a case-sensitive value. It is case-sensitive because it can be used as a value in one of the document hierarchies to locate an object.

The example should present the word "EXAMPLE" on all compliant browsers.

Warnings:

- This value is read/write in MSIE, but read-only in Netscape. Logically there is not much purpose in changing the name of an `anchor` anyway.
- Beware that assigning a new name will affect the length of the `document.anchors[]`.
- Changing the name in MSIE actually adds a new item to the `document.anchors[]` array that can be reached associatively with the new name. There will now be two entries for the same anchor and you can continue to access it using the old name as well.
- If you are writing portable code and expect it to work in both MSIE and Netscape Navigator, this is the only property available in both browsers. Having located an anchor, being able to access only its name without any browser dependencies is rather limiting.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE" HREF="http://www.mydomain.com/folder/file.html#abcdef">Click
here</A>
<BR>
<SCRIPT>
document.write(document.anchors[0].name);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor` object, `AnchorArray.length`, `Document.anchors[]`, `NAME="..."`, `String.anchor()`, `Url.name`

Anchor.nameProp (Property)

The filename portion of the URL value.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myAnchor*`.nameProp`

This property extracts the filename portion of the `HREF` value for this `<A>` tag.

See also:	`Url.nameProp`

Property attributes:

`ReadOnly`.

Anchor.pathname (Property)

In MSIE the `Url.pathname` property is also available as the `Anchor.pathname` property.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.pathname`
	-	*myAnchor*`.pathname = `*newPath*
HTML syntax:	`<A HREF="...">`	

This yields the pathname portion of the `HREF` attribute in an `<A>` tag.

MSIE and Netscape support the use of this property as an LValue. If you write to it, the pathname portion of the `HREF` value is modified. Be careful not to include a hash or search/query value.

Warnings:

- Since the `pathname` property of an `Anchor` object is not portable in all older browser versions, you should use the `pathname` property of the corresponding `Url` object to be able to work across MSIE and Netscape.
- As features become deprecated, it may become necessary to support both techniques in browser-specific code according to your needs regarding the support of legacy browsers.

See also:	`Anchor` object, `Anchor.hash`, `Anchor.host`, `Anchor.hostname`, `Anchor.href`, `Anchor.port`, `Anchor.protocol`, `Anchor.search`, `Anchor.target`, URL, `Url.pathname`

Anchor.port (Property)

In MSIE the `Url.port` property is also available as the `Anchor.port` property.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.port`
	-	*myAnchor*`.port = `*newPort*
HTML syntax:	`<A HREF="...">`	

This yields the port number value of the HREF attribute in an <A> tag.

You can assign a value to this property as if it were an LValue.

Warnings:

- Since the port property of an Anchor object is not portable, you should use the port property of the corresponding Url object to be able to work across MSIE and Netscape.
- Do not include the delimiting colon when you assign a value to this property.
- Make sure you assign a numeric value. Non-numeric values will be rejected to avoid the possibility of a completely invalid port number being used.

See also:	Anchor object, Anchor.hash, Anchor.host, Anchor.hostname, Anchor.href, Anchor.pathname, Anchor.protocol, Anchor.search, Anchor.target, URL, Url.port

Anchor.protocol (Property)

In MSIE the Url.protocol property is also available as the Anchor.protocol property.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	String primitive
JavaScript syntax:	- *myAnchor*.protocol - *myAnchor*.protocol = *newProtocol*
HTML syntax:	<A HREF="...">

This yields the protocol value of the HREF attribute in an <A> tag.

Using this property as an LValue, you can redefine the protocol for the link if it has an HREF. You might want to do this if you want to change the way you access a particular document.

The URL topic enumerates a large number of available protocols that can be used in SRC and HREF HTML tag attributes.

Warnings:

- Since the protocol property of an Anchor object is not portable, you should use the protocol property of the corresponding Url object to be able to work across MSIE and Netscape.

See also:	Anchor object, Anchor.hash, Anchor.host, Anchor.hostname, Anchor.href, Anchor.pathname, Anchor.port, Anchor.search, Anchor.target, IMG.protocol, URL, Url.protocol

Anchor.protocolLong (Property)

A long form description of the protocol used by the URL.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myAnchor*`.protocolLong`

Only the MSIE browser supports this property. Its use would be limited even if it were available across multiple platforms.

See also:	`Url.protocolLong`

Property attributes:

`ReadOnly`.

Anchor.recordNumber (Property)

The record within the data set that defined the element content when the content came from a data source.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myBody*`.recordNumber`

This is a property that is part of the MSIE data-binding support. It contains an integer value that is the record number within the data set that created this object.

This is useful when you are building pages with Active Server Pages (ASP) and ActiveX Data Objects (ADO).

Property attributes:

`ReadOnly`.

Anchor.rel (Property)

A definition of the relationship between the current document and the document at the location specified by the URL.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*.rel

This is sometimes called a forward link. Although the `HREF` HTML tag attribute is normally the only means used to identify a target document, the browser is permitted to use the `REL` HTML tag attribute to decide whether to use the `HREF` value or how it should be used.

The following HTML version 4.0 standard link types are permitted in this property:

- `alternate`
- `appendix`
- `bookmark`
- `chapter`
- `contents`
- `copyright`
- `glossary`
- `help`
- `index`
- `next`
- `prev`
- `section`
- `start`
- `stylesheet`
- `subsection`

MSIE adds these as well:

- `same`
- `next`
- `parent`
- `previous`

When used or tested within a script, any comparisons should be case-insensitive.

See also:	`Anchor.rev`, `LINK.rel`

Anchor.rev (Property)

A complementary description of the link to the current document as viewed from the document at the location specified by the URL.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*.rev

This is sometimes called a reverse link. It defines the relationship between a document and another that calls it. The linkage is defined from the destination document's viewpoint.

This property supports the same HTML version 4.0 standard link types as the `rel` property. Refer to that topic for details.

When used or tested within a script, any comparisons should be case-insensitive.

Because `rel` and `rev` properties are complementary, the values in them are likely to be related. For example, if one contains the value "next" then the other is likely to contain "previous".

Refer to the Anchor.rel topic for a list of the available types you can use in this property.

See also:	`Anchor.rel`, `LINK.rev`

Anchor.search (Property)

The query portion of an `HREF` URL if there is one.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*.search
	-	*myAnchor*.search = *newSearch*
HTML syntax:	`<A HREF="...">`	

Warnings:

- Since the `search` property of an `Anchor` object is not portable, you should use the `search` property of the corresponding link object to be able to work across MSIE and Netscape.

See also:	`Anchor` object, `Anchor.hash`, `Anchor.host`, `Anchor.hostname`, `Anchor.href`, `Anchor.pathname`, `Anchor.port`, `Anchor.protocol`, `Anchor.target`, `request.<urlExtension>`, URL, `Url.search`

Anchor.shape (Property)

A map whose extent is defined by the `coords` property and which can be one of several different shapes.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*`.shape`

This property has a meaningful value when the `Anchor` object is instantiated via `<MAP>` and `<AREA>` tag. It defines the shape of the hotspot within the extent rectangle defined by the `coords` property. It might contain one of the following values:

- `default`
- `rect`
- `circle`
- `poly`

See also:	`Anchor.coords`, `Area.shape`, `Url.shape`

Anchor.tabIndex (Property)

A control of where the `Anchor` object appears in the tabbing order of the page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myInputObject*`.tabIndex`

This value indicates where in the tabbing sequence this object and any of its children will be placed. The tabbing order is used when filling in forms or moving focus. Pressing the [*tab*] key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

Anchor.target (Property)

In MSIE the `Url.target` property is also available as the `Anchor.target` property.

Availability:	DOM level – 1 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myAnchor*`.target` - *myAnchor*`.target = `*newTarget*
HTML syntax:	`<A TARGET="...">`

This yields the value of the `TARGET` attribute in an `<A>`, `<AREA>`, or `<MAP>` tag.

You can assign a new value to this property so that the URL will be directed to a different window or frame.

Here are some example target values:

- `_parent`
- `_self`
- `_top`
- `_blank`
- Window name
- Frame name

Warnings:

- Since the `target` property of an `Anchor` object is not portable, you should use the `target` property of the corresponding `link` object to be able to work across MSIE and Netscape.

See also:	`<MAP TARGET="...">`, `Anchor` object, `Anchor.hash`, `Anchor.host`, `Anchor.hostname`, `Anchor.href`, `Anchor.pathname`, `Anchor.port`, `Anchor.protocol`, `Anchor.search`, `BASE.target`, `Form.target`, `Location.target`, `Map.target`, URL, `Url.target`

Anchor.text (Property)

The text between the <A> and </A> HTML tags in Netscape.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myAnchor*.text
	N	*myAnchor*.text = *aString*
HTML syntax:	<A>someText</A>	
Argument list:	*aString*	Some new text content for the anchor

This is equivalent to the `innerText` value that MSIE supports. It only works on Netscape and is somewhat less reliable than the `innerText` property in MSIE.

Assigning to this property in MSIE simply creates a text property, but does not affect the text of the anchor.

The value yielded by this property (when it does work) is the text between the <A> and </A> tags.

Warnings:

- You will need to detect the browser type before attempting to use this property.
- Does not work on Netscape Navigator version 4.7 on the Macintosh. Instead it displays some fragment of body text that comes from outside the anchor tags.
- Even if it does work, you may only extract a portion of the text from the anchor.

See also:	`Anchor` object, `Element.innerText`, `Url.text`

Property attributes:

`ReadOnly`.

Anchor.type (Property)

A MIME type value in its abbreviated machine recognizable form.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myAnchor*.type

The MIME type of the document associated with the `Anchor` object is accessible through the value of this property.

Refer to the MIME type topic for details of the available MIME types you will likely see in this property.

See also:	`LINK.type`, MIME types

Anchor.urn (Property)

An alternative format of the contents of the URL.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myAnchor*`.urn`

See also:	URN

Anchor.x (Property)

The X location of the anchor within the document.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myAnchor*`.x`
	N	*myAnchor*`.x = `*aValue*
Argument list:	*aValue*	A new X coordinate value

The `Anchor.x` property yields the pixel distance of the anchor from the left edge of the document. The horizontal position of the object in the display is measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Warnings:

- This is not supported by MSIE. Instead you can use the `offsetLeft` property that is inherited from the `Element` object super-class. There may be some occasions when this is not an exact equivalent value though.

See also:	`Anchor` object, `Element.offsetLeft`, `Location.x`

Property attributes:

`ReadOnly`.

Anchor.y (Property)

The Y location of the anchor within the document.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myAnchor*`.y`
	N	*myAnchor*`.y = `*aValue*
Argument list:	*aValue*	A new Y coordinate value

The `Anchor.y` property yields the pixel distance of the anchor from the top edge of the document. The vertical position of the object in the display is measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Warnings:

- This is not supported by MSIE. Instead you can use the `offsetTop` property that is inherited from the `Element` object super-class. There may be some occasions when this is not an exact equivalent value though.

See also:	`Anchor` object, `Element.offsetTop`, `Location.y`

Property attributes:

`ReadOnly`.

AnchorArray object (Object/DOM)

An array of `Anchor` objects retrieved from the `document.anchors` property.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 3.0 Internet Explorer – 4.0 Netscape – 2.0	
JavaScript syntax:	-	*myAnchorArray* `= `*myDocument*`.anchors`
Object properties:	`length`	

The `AnchorArray` object is a sub-class of the `Array` object but has no additional properties. It responds to the `length` property request as you would expect.

Any `Anchor` objects in this array can be accessed by index value because the `Array` class supports that. In Netscape, the individual `Anchor` objects are accessible associatively by their `NAME` attribute. However, MSIE does not make this associative mechanism available.

In MSIE, the `AnchorArray` object is a kind of `Collection` object and so it can be searched with the `item()` and `tags()` methods.

Warnings:

- Netscape adds a `constructor` property to this object class from which you can request the name to determine the object class. Actually Netscape provides constructors for virtually everything, but MSIE only supports them when it's necessary and useful.
- Be aware that renaming an anchor in MSIE will add a new item to the `AnchorArray` collection without destroying the old one. However, the `length` property remains the same. It does mean that you could have problems enumerating the collection. But then, why would you ever want to rename an anchor after it has been instantiated and named within the HTML tag?

Example code:

```
<!-- Catalog of anchors in an array -->
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="A1" HREF="http://www.apple.com/">Apple</A><BR>
<A NAME="A2" HREF="http://www.wrox.com/">Wrox</A><BR>
<A NAME="A3" HREF="http://www.msdn.com/">Microsoft</A><BR>
<BR>
<HR>
<TABLE BORDER=1>
<TH>Index</TH>
<TH>Name</TH>
<TH>Text</TH>
<TH>URL</TH>
<TH>Tab index</TH>
<TH>Protocol (long)</TH>
<SCRIPT>
myLength = document.anchors.length;
for (myEnumerator=0; myEnumerator<myLength; myEnumerator++ )
{
document.write("<TR><TD>");
document.write(myEnumerator);
document.write("</TD><TD>");
document.write(document.anchors[myEnumerator].name);
document.write("</TD><TD>");
document.write(document.anchors[myEnumerator].innerText);
document.write("</TD><TD>");
document.write(document.anchors[myEnumerator].href);
document.write("</TD><TD>");
```

```
document.write(document.anchors[myEnumerator].tabIndex);
document.write("</TD><TD>");
document.write(document.anchors[myEnumerator].protocolLong);
document.write("</TD></TR>");
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	`Anchor` object, `Collection` object, `Document.anchors[]`

Property	**JavaScript**	**JScript**	**N**	**IE**	**Opera**	**DOM**	**Notes**
`length`	1.0 +	3.0 +	2.0 +	4.0 +	-	-	ReadOnly

AnchorArray.length (Property)

The number of named anchors in the current document.

Availability:	JavaScript – 1.0 JScript – 3.0 Internet Explorer – 4.0 Netscape – 2.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDocument*`.anchors.length`

The length of the `anchors` array, which indicates the number of named `<A>` HTML tags in the document.

See also:	`Anchor.name`, `Collection.length`, `Document.anchors[]`

Property attributes:

`ReadOnly`.

Anonymous code (Definition)

Script source executed by function objects.

Availability:	ECMAScript edition – 2

Anonymous code is script source text that is supplied to the constructor when an anonymous function is being instantiated.

As the constructor creates the function, one of the arguments to the constructor call will contain the script source text to be stored as the code block associated with that function. Only the code in the last argument on the constructor call is used in this way. All but the last argument are concatenated together separated by commas and are then passed as the formal parameter list to the anonymous function.

Anonymous code initializes the scope chain to include its `activation` object followed by the `global` object.

Variable instantiation is performed using the `activation` object as the variable object and any initial variables are flagged with a `DontDelete` attribute.

The caller provides this value, but in some situations the value null may be passed. In that case, the `global` object will be used in its place.

If you need more information, the ECMA standard is the authoritative source on what anonymous code is and how a compliant interpreter implementation should manage it.

See also:	Executable code, Execution context

Cross-references:

ECMA 262 edition 2 – section – 10.1.2

ECMA 262 edition 2 – section – 10.1.6

ECMA 262 edition 2 – section – 10.2.3

ECMA 262 edition 3 – section – 10.1.2

ECMA 262 edition 3 – section – 10.1.6

ECMA 262 edition 3 – section – 10.2.3

Anonymous function (Definition)

A manually constructed function object with no identifying name.

Availability:	ECMAScript edition – 2

Anonymous functions are created dynamically by using the built-in `Function` object as a constructor in a function expression.

As they are called, the last argument is used to provide the script source, and all but the last argument are used as a formal parameter list for the function.

Anonymous functions are properly supported by the WebTV set-top box from the Summer 2000 release onwards. Earlier versions of this product only partially supported anonymous functions.

See also:	Function literal, `Function` object, Instantiating Function, JellyScript

Cross-references:

ECMA 262 edition 2 – section – 10.1.1

ECMA 262 edition 3 – section – 10.1.1

Applet object (Object/HTML)

An object representing an HTML `<APPLET>` tag.

Availability:	DOM level – 0 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0 Deprecated – HTML 4.0, DOM level 1
Inherits from:	`Element` object
JavaScript syntax:	- *myApplet* = *aName* - *myApplet* = *myAppletArray*[*aName*] - *myApplet* = *myAppletArray*[*anIndex*] IE *myApplet* = *myDocument*.all.*anElementID* IE *myApplet* = *myDocument*.all.tags("APPLET")[*anIndex*] IE *myApplet* = *myDocument*.all[*aName*] - *myApplet* = *myDocument*.*aName* - *myApplet* = *myDocument*.applets[*aName*] - *myApplet* = *myDocument*.applets[*anIndex*] - *myApplet* = *myDocument*.getElementById(*anElementID*) - *myApplet* = *myDocument*.getElementsByName (*aName*)[*anIndex*] - *myApplet* = *myDocument*.getElementsByTagName ("APPLET")[*anIndex*]
HTML syntax:	`<APPLET> ... </APPLET>`
Argument list:	*aName* – The name of an applet *anIndex* – An element in the `applets` collection *anElementID* – The ID value of an `Element` object
Object properties:	`accessKey`, `align`, `alt`, `altHTML`, `archive`, `code`, `codeBase`, `dataFld`, `dataSrc`, `form`, `height`, `hspace`, `name`, `object`, `src`, `tabIndex`, `vspace`, `width`
Object methods:	`start()`, `stop()`
Event handlers:	`onAfterUpdate`, `onBeforeUpdate`, `onBlur`, `onClick`, `onDataAvailable`, `onDataSetChanged`, `onDataSetComplete`, `onDblClick`, `onErrorUpdate`, `onFocus`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onLoad`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onReadyStateChange`, `onResize`, `onRowEnter`, `onRowExit`

The properties and methods of an `Applet` object are inherited from the public properties and methods of the `Java` object it represents. However, in addition to these, MSIE also supports some additional properties.

The Java applet itself is the concrete object whose properties are accessed.

In Netscape, Applets are encapsulated as instances of the `JavaObject` class and communicate by means of the LiveConnect support. The mechanisms are quire different in MSIE, which uses ActiveX facilities to access applets.

When you access an `Applet` (`JavaObject`) object, you are really interacting with the Java applet itself.

The publicly accessible properties and methods depend on the applet, although all applets must support the `start()` and `stop()` methods.

It is generally safer to interact with methods that you have provided as custom additions to the applet, rather than hope that the applet supports any particular methods.

Because Java is so much more strongly data-typed than JavaScript, you must be careful with the kind of values you try and send to and receive from a Java applet. Java will also not forgive the omission of an argument. In JavaScript, all arguments are assumed to be optional as a general rule, although leaving them out will have strange side effects sometimes. Java will not allow you to do this and a run-time error will be generated if the arguments are not complete and all of the correct type.

In Netscape, you can build an enumerator loop to examine all the properties of an `Applet` object. Enumerating applet interfaces like this will yield a long list of function objects. Each function object represents an accessor for internal properties of the Java environment. Your applet may publish additional properties. With these functions, you can enquire about certain attributes of the applet and can change some of them from the script. Refer to the `JavaObject` topic for details about these generic capabilities, but bear in mind they only work in Netscape.

In MSIE, the `APPLET` object inherits its behavior from the `Element` object. Refer to the topic covering that for its generic properties and methods. MSIE supports many other properties and methods that are not generally available to `Element` objects and these are detailed here as properties and methods of the `Applet` object.

Warnings:

- MSIE implements this object as a member of the class `APPLET` rather than `Applet` as you would expect.
- Netscape implements it as a member of the class `JavaObject`, although this is masked by some shortcomings in the implementation that prevent it from displaying its class type.
- `<APPLET>` tags are deprecated in HTML 4.0 and DOM level 1, which suggests there may be some changes to the JavaScript support for them in subsequent implementations of JavaScript in browsers.

See also:	ActiveX, `Applet.start()`, `Applet.stop()`, `AppletArray` object, `Document.applets[]`, `Element` object, `Input.accessKey`, `JavaObject` object, LiveConnect

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	0 +	-	Warning, Deprecated
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
alt	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
altHTML	-	3.0 +	-	4.0 +	-	-	-	-
archive	1.1 +	5.0 +	3.0 +	5.0 +	-	1 +	-	-
code	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
codeBase	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
dataFld	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	0 +	-	Warning, Deprecated
dataSrc	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	0 +	-	Warning, Deprecated
form	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	0 +	-	Warning, Deprecated
height	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
hspace	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
name	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	ReadOnl.
object	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
src	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
tabIndex	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	0 +	-	Warning, Deprecated
vspace	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
start()	1.1 +	3.0 +	3.0 +	4.0 +	-	-	-	-
stop()	1.1 +	3.0 +	3.0 +	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onClick	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDataAvailable	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetChanged	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetComplete	-	3.0 +	-	4.0 +	-	-	-	-
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onLoad	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onReadyStateChange	-	3.0 +	-	4.0 +	-	-	-	-
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Applet() (Constructor)

You normally would not use the constructor to create new applets, but it is possible to do this if you need to create an applet container.

Availability	`Deprecated`
Property/method value type:	`Applet` object

As is the case with many (but not all) objects in Netscape, you can call a constructor to create a new instance of an object. MSIE does not generally support this unless a constructor is really justified. Because this constructor is only supported in Netscape, you should avoid constructing new `Applet` objects. In any case, they are of limited use since you cannot easily place them into the page and make them visible, even if you could populate them with meaningful content. Because of this, the topic is marked as deprecated, although the functionality is likely to continue to be available.

Applet.align (Property)

Determines how the applet area aligns with its surrounding content.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.align`

The alignment of the applet with respect to its containing parent object is defined in this property. The following expected set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

Applet.alt (Property)

The alternative text to be used instead of the applet block in case the applet fails to load or for use as a tool-tip text.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.alt`

If a browser loads a document containing an applet and discovers that it is unable to load the applet, this text value of this property will be displayed in the space where the applet was supposed to have been loaded.

The use of this property is somewhat problematical in some browsers and completely unsupported in others.

Setting this property from a script is unlikely to be very useful, as the script is probably going to be called after the applet has failed to load.

Applet.altHTML (Property)

Some alternative HTML to display if the applet fails to load.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myApplet`*`.altHTML`

If a browser loads a document containing an applet and discovers that it is unable to load the applet, this HTML will be displayed in the space where the applet was supposed to have been loaded.

However, this property cannot be set from an HTML tag attribute as the `alt` text can. It can only be set from a script.

Setting this property from a script is unlikely to be very useful, as the script is probably going to be called after the applet has failed to load.

Applet.archive (Property)

The name of a zip archive containing multiple class files.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 5.0 Internet Explorer – 5.0 Netscape – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myApplet*.archive

Netscape allows for multiple applet class files to be collected together into a single zipped archive file. Useful performance gains are possible if an applet depends on several classes for its implementation, because they can all be loaded at once.

For this to work, you must also specify the `CODE` HTML tag attribute so that the browser can determine which one of the classes is the main one.

There is some variance here from the HTML 4.0 definition of this value, which suggests that a list of space-separated URL values can be specified. That is intended for use with the `<OBJECT>` tag, which, according to the W3C organization, is the successor to the `<APPLET>` tag, which is likely to be come deprecated in due course. However, since the `<APPLET>` tag is used so widely, this is likely to take some considerable time to take effect.

See also:	`Applet.code`

Applet.code (Property)

The Java class code for the applet.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myApplet*.code

This specifies the main code class to be called when the applet is initially run. It is necessary to identify the main item in case there was a collection of class files loaded as an archive.

See also:	`Applet.archive`

Property attributes:

`ReadOnly`.

Applet.codeBase (Property)

The path to the directory containing the applet code.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.codeBase`

The codebase is the path to the directory where the classes used in the code or archive properties are located. The actual path to the required files is generated by a string concatenation of `codeBase+code` or `codeBase+archive` to generate a fully specified URL.

Due to security limitations, it is not permitted to access a codebase value that is outside the domain specified by the containing document.

Property attributes:

`ReadOnly`.

Applet.height (Property)

The height of the applet extent rectangle in pixels.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.height`

The extent rectangle around the applet reserves some space in the display before the applet is loaded. The height of that extent rectangle is specified in this property and is normally measured in pixels. Length values controlled by CSS styles allow for sizes to be specified in other measurement units.

See also:	`Applet.width`

Property attributes:

`ReadOnly`.

Applet.hspace (Property)

The width of the horizontal margin spacing around an `Applet` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.hspace`

Margins placed around objects are either modified separately with all four margin sides having a different property or by adjusting the horizontal margins and vertical margins using just two values.

The `hspace` property controls the margin to the left and right of the object.

Applet.name (Property)

This corresponds to the `NAME` attribute of the `<APPLET>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.name`
HTML syntax:	`<APPLET NAME="aName">`	
Argument list:	*`aName`*	A name to identify the `Applet` object

Objects are identified either by the `NAME` HTML tag attribute or by the `ID` HTML tag attribute.

Netscape shows a marginal preference for the `name` property while MSIE seems slightly better disposed towards the `ID` property. However, in many cases both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

Property attributes:

`ReadOnly`.

Applet.object (Property)

An accessor that yields a reference to the containing JavaScript object when there is a possibility of naming conflicts between internally visible and externally visible property names.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Applet` object	
JavaScript syntax:	-	*myApplet*`.object`

There are occasional namespace conflicts when using applets. Public properties are created in a different environment, but are published into the JavaScript namespace and take precedence over the properties of the containing JavaScript object.

The problem is exhibited when the name of a public property collides with a property of the containing JavaScript object instantiated by the `<APPLET>` HTML tag. Access to the property belonging to the containing object is difficult because the scope search order will obtain the public property of the applet first. By using the `object` property, you can access the containing object explicitly and retrieve a property of that object even if there is an identically named property belonging to the enclosed `Applet` object.

This access mechanism applies to method invocations as well.

See also:	`OBJECT.object`

Applet.src (Property)

A supplementary property for passing in URL values to the applet.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myApplet*`.src`

Some applets may need to access a supplementary data file from the server. It is good practice to abstract such data values from the code itself, and so a means of passing this parameter value in from outside is necessary. The `SRC` HTML tag attribute is reflected into this property and is provided as a somewhat standardized means of passing one of the parameter values most likely to be defined.

Property attributes:

`ReadOnly`.

Applet.start() (Method)

A public method that starts an applet running.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
JavaScript syntax:	-	*myApplet*`.start()`

This method will start an applet running if it has previously been stopped. Note that, in general, applets will run automatically by default unless you do something to prevent it (possibly by setting HTML tag attributes).

See also:	`Applet` object, `Applet.stop()`

Applet.stop() (Method)

A public method that stops an applet running.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
JavaScript syntax:	-	*myApplet*`.stop()`

This method provides a way to stop the execution of an applet from outside. Applets may choose to stop themselves if that is what you have designed them to do. Other applets may be embedded into the page and instructed not to run automatically by setting the appropriate attributes in the `<APPLET>` HTML tag.

See also:	`Applet` object, `Applet.start()`

Applet.vspace (Property)

The height of the vertical margin spacing around an `Applet` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.vspace`

Margins placed around objects are modified either separately with all four margin sides having a different property or by adjusting the horizontal margins and vertical margins using just two values.

The `vspace` property controls the margin at the top and bottom of the object.

Applet.width (Property)

The width of the applet extent rectangle in pixels.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myApplet`*`.width`

The extent rectangle around the applet reserves some space in the display before the applet is loaded. The width of that extent rectangle is specified in this property and is normally measured in pixels. Length values controlled by CSS styles allow for sizes to be specified in other measurement units.

See also:	`Applet.height`

Property attributes:

`ReadOnly`.

AppletArray object (Object/DOM)

A sub-class of the `Array` object that implements an applet collection.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0	
JavaScript syntax:	-	*myAppletArray* = *myDocument*.applets
Object properties:	length	

Warnings:

- Although Netscape supports a constructor for this object type, it appears to point at the wrong thing. In any case, it's unlikely you'd want to create a new `AppletArray`.

See also:	`Applet` object, `AppletArray.length`, `Collection` object, `Document.applets[]`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
length	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	1 +	-	-	ReadOnly

AppletArray.length (Property)

The number of `Applet` objects in the collection.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myDocument*.applets.length

The length of the `applets` array, which indicates the number of `<APPLET>` HTML tags in the document.

See also:	`AppletArray` object, `Collection.length`

Property attributes:

`ReadOnly`.

Area object (Object/HTML)

An object representing an `<AREA>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myArea* = *myDocument*.all.*aMapID*.areas[*anIndex*]
	IE	*myArea* = *myDocument*.all.*anElementID*
	IE	*myArea* = *myDocument*.all.tags("AREA")[*anIndex*]
	IE	*myArea* = *myDocument*.all[*aName*]
	-	*myArea* = *myDocument*.getElementById (*anElementID*)
	-	*myArea* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myArea* = *myDocument*.getElementsByTagName ("AREA")[*anIndex*]
	-	*myArea* = *myDocument*.links[*anIndex*]
HTML syntax:	`<AREA>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	accessKey, alt, coords, hash, host, hostname, href, name, noHref, pathname, port, protocol, search, shape, tabIndex, target, text, x, y	
Object methods:	add()	
Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onClick, onDataAvailable, onDataSetChanged, onDataSetComplete, onDblClick, onErrorUpdate, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReadyStateChange, onResize, onRowEnter, onRowExit	

An `Area` object represents an area of an image map. They are generally referred to as `Link` objects, although Netscape and MSIE instantiate them as different classes.

Netscape supports these objects as objects of the `Url` class.

MSIE treats them as `Link` objects.

Event-handling support via properties containing function objects was added to `Area` objects at version 1.1 of JavaScript.

See also:	`Element` object, `HyperLink` object, `Input.accessKey`, `LINK` object, `LinkArray` object, `Location` object, `Map` object, `Url` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`accessKey`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`alt`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`coords`	1.5 +	3.0 +	6.0 +	3.02 +	-	1 +	-	-
`hash`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
`host`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
`hostname`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
`href`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	1 +	-	-
`name`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	-	-
`noHref`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`pathname`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
`port`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
`protocol`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
`search`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
`shape`	1.5 +	1.0 +	6.0 +	3.02 +	-	1 +	-	-
`tabIndex`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`target`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	1 +	-	-
`text`	1.2 +	-	4.0 +	-	-	-	-	-
`x`	1.2 +	-	4.0 +	-	-	-	-	-
`y`	1.2 +	-	4.0 +	-	-	-	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`add()`	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onClick`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	4.0 +	Warning
`onDataAvailable`	-	3.0 +	-	4.0 +	-	-	-	-
`onDataSetChanged`	-	3.0 +	-	4.0 +	-	-	-	-
`onDataSetComplete`	-	3.0 +	-	4.0 +	-	-	-	-
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onErrorUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onLoad	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onReadyStateChange	-	3.0 +	-	4.0 +	-	-	-	-
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Area.accessKey (Property)

A key that needs to be pressed before the mapped link will respond to data entry.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*.accessKey

The key defined in this property needs to be held down for any input events to be triggered on this object or its children.

Area.add() (Method)

Add a new element to the `Area` object that describes the image map.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myArea*.add(*anObject*)
	IE	*myArea*.add(*anObject*, *anIndex*)
Argument list:	*anObject*	A new `link` object to add
	anIndex	A position in the collection to add the new item

Image maps can be modified from the scripting interface. You might find this useful if you present some new information and want to add a button to dismiss it. It is possible to avoid an unnecessary screen redraw and image load by adding an item to an image map collection.

Area.alt (Property)

The tool-tip text for the `Area` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.alt`

Objects can have an alternative text string associated with them. Browsers that cannot cope with the tag may display the text instead. If spoken styles are supported, the text may be read out to the user. Some browsers will also display the alt text as a tool-tip if the mouse is positioned over the object and remains static for a few seconds.

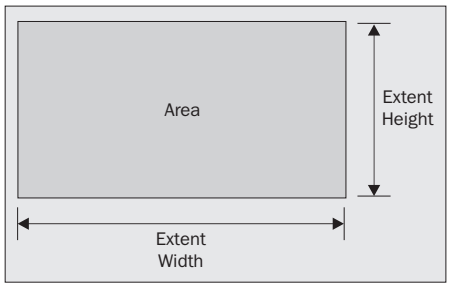
Area.coords (Property)

The extent rectangle for the `Area` object within a map.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 3.02 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.coords`

When a shaped area is defined within a an image map, the extent rectangle around the shape is defined with the `coords` property. The value is defined with the `COORDS` HTML tag attribute.

See also:	`Anchor.coords`, `Area.shape`



Area.hash (Property)

MSIE represents URLs in `Link` objects.

Availability:	JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.hash`

See also:	`Url.hash`

Area.host (Property)

MSIE represents URLs in `Link` objects.

Availability:	JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.host`

See also:	`Url.host`

Area.hostname (Property)

MSIE represents URLs in `Link` objects.

Availability:	JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.hostname`

See also:	`Url.hostname`

Area.href (Property)

MSIE represents URLs in `Link` objects.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myArea*.href
HTML syntax:	`<AREA HREF="...">`

See also:	`Location.href`, `Url.href`

Area.name (Property)

This corresponds to the `NAME` attribute of the `<AREA>` HTML tag.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myArea*.name

Objects are identified either by the `NAME` HTML tag attribute or by the `ID` HTML tag attribute.

Netscape shows a marginal preference for the `name` property, while MSIE seems slightly better disposed towards the `ID` property. However, in many cases both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

See also:	`Url.name`

Area.noHref (Property)

A Boolean flag to indicate whether the area is a link or a dead spot within the map.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myArea*.noHref

When a shaped area is defined within a an image map, it can either define a live hotspot or a hole that has been cut out in the map. In this way, both concave and convex shapes can be created. You can also create shapes with holes in the middle.

Area.pathname (Property)

MSIE represents URLs in `Link` objects.

Availability:	JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*.pathname
See also:	`Url.pathname`	

Area.port (Property)

MSIE represents URLs in `Link` objects.

Availability:	JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myArea*.port
See also:	`Url.port`	

Area.protocol (Property)

MSIE represents URLs in `Link` objects.

Availability:	JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.protocol`

See also:	`IMG.protocol`, URL, `Url.protocol`

Area.search (Property)

MSIE represents URLs in `Link` objects.

Availability:	JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.search`

See also:	`request.<urlExtension>`, `Url.search`

Area.shape (Property)

The shape of the extent area within the map.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 1.0 Internet Explorer – 3.02 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArea*`.shape`

This property has a meaningful value when the `Area` object is instantiated via `<MAP>` and `<AREA>` tag. It defines the shape of the hotspot within the extent rectangle defined by the `coords` property. It might contain one of the following values:

- `default`
- `rect`
- `circle`
- `poly`

See also:	`Anchor.shape,Area.coords,Url.shape`

Area.tabIndex (Property)

A control of where the `Area` object appears in the tabbing order of the page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myArea*`.tabIndex`

This value indicates where this object and any of its children will be placed in the tabbing sequence. The tabbing order is used when filling in forms or moving focus. Pressing the [*tab*] key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

Area.target (Property)

MSIE represents URLs in `Link` objects.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myArea*`.target`
HTML syntax:	`<AREA TARGET="...">`
See also:	`<MAP TARGET="...">,Anchor.target,BASE.target, Form.target,Location.target,Map.target, Url.target`

Area.text (Property)

Netscape represents `<AREA>` tags as `Url` objects and therefore they inherit this property.

Availability:	JavaScript – 1.2 Netscape – 4.0
Property/method value type:	`String primitive`
JavaScript syntax:	N *myArea*`.text`
See also:	`Url.text`

Area.x (Property)

Netscape provides this as an enumerable property because it represents an `<AREA>` as a `Url` object.

Availability:	JavaScript – 1.2 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	N *myArea*`.x`

The horizontal position of the object in the display measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

See also:	`Location.x`

Area.y (Property)

Netscape provides this as an enumerable property because it represents an `<AREA>` as a `Url` object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myArea*`.y`

The vertical position of the object in the display measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

See also:	`Location.y`

areas[] (Collection)

A collection of all the `Area` objects that contribute to making an image map for the page.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myMap*`.areas`

See also:	`Map.areas[]`

Property attributes:

`ReadOnly`.

argc parameter (Definition)

A command-line argument count.

Since JavaScript can be used in many environments, it is possible that in a server-side application you will have access to the command-line arguments.

If that is the case, then it is likely that you will have an `argc` property, which indicates how many arguments have been passed.

In general, the first argument is the name of the script or program being executed. The `argc` value should never be zero and as a minimum should indicate that there is at least one argument.

The actual values of the arguments are collected in an array called `argv`. You should be able to access `argv` and `argc` in a similar manner.

See also:	`argv` parameter, Execution context, Execution environment, Host environment, `main()` function

Argument (Definition)

A value passed to a function.

Arguments are passed to functions when they are called. They are substituted for the formal parameters in the function declaration.

Because JavaScript is weakly typed, you will need to implement any type checking you need for yourself.

You can compare the `arity` property of the owning function with the `length` property of the `arguments` array. If they are unequal, then the function was called with the wrong number of arguments.

You can then check the type of the arguments one by one to compare them against the expected types.

This is a lot of work for little gain unless it is an important aspect of your design.

See also:	`Arguments` object, `Arguments.length`, Conversion, Definition, Function, `function(…) …`, `Function.arguments[]`, Parameter

Argument list (Definition)

A list of values that are passed to a function.

Availability:	ECMAScript edition – 2

Argument lists are used to pass information into functions.

An argument list can have any of the following structures:

- Empty – no arguments
- A single argument
- A series of arguments separated by commas

Each argument, if present, can be an expression that will be evaluated and whose resulting value will be used as the argument when it is passed to the function.

See also:	`Arguments` object, `Arguments.length`, `Function.arguments[]`, Left-Hand-Side expression, Parameter

Cross-references:

ECMA 262 edition 2 – section – 11.2.4

ECMA 262 edition 3 – section – 11.2.4

Arguments object (Object/core)

An object represented as an array containing the argument values passed to the function when it is called.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 5.5 Internet Explorer – 5.5 Netscape – 3.0	
JavaScript syntax:	-	*myArguments* = arguments
Object properties:	callee, caller, length	

When you call a function, you can pass zero or more arguments to it from outside. These arguments are available as named variables whose names are defined in the function declaration.

However, they are also available as the elements in an array. The `arguments` array is referenced by the `arguments` property of the `call` object. Since the `call` object is added to the scope chain, you don't need to reference the `arguments` property with an object identifier prefix.

The array-based mechanism is useful for those times when you want to implement a function that has a variable number of arguments passed to it according to how and when it is called.

A new `arguments` object is created for each execution context. When the flow of control enters an execution context for a function block, a new `arguments` object is created. Declared functions, anonymous code, and implementation-specific code all use this technique.

When creating the `arguments` object, the initial conditions are set up like this:

- The internal `Prototype` property for the `arguments` object is that returned by calling `Object.Prototype`.
- A property is created with the name `callee`. The `callee` property cannot be enumerated. The initial value of the `callee` property is the `function` object being executed. Anonymous functions can then be executed recursively if you so desire.
- A property named `length` is created whose value is the number of arguments passed to the function. The `length` property cannot be enumerated.
- Each argument is associated with a property whose name is its integer position in an array of arguments. The arguments are accessed in presentation order. Although the names are strings, they represent purely numeric values and range from 0 to 1 less than the value in the `length` property. You can enumerate the arguments in a `for` loop.

Note that objects of this type can only exist within a function body in a web browser, because you cannot pass parameters to a script from outside. It is possible that an embedded JavaScript interpreter may provide a `host` object to the main entry point to perform the same function.

Warnings:

- In Netscape, the `arguments` array is implemented as an object of type `Arguments` but in MSIE its type is simply an `Object` object. In Netscape, the `arguments` object is extended with a `toString()` mechanism that returns the arguments as a comma separated list in a String. In MSIE, you get the object type.
- None of the properties of the `arguments` object are enumerable.
- Because the `arguments` object is meant to be used in a manner that is local to the function it was created in, you get unpredictable results if you pass it to another function as an argument itself.
- Note that at the time of writing the example given below did not seem to work on Netscape 6.0.

Example code:

```
<HTML>
<BODY>
<SCRIPT>
// Call a function and use its arguments array find out the
// name of the function that called it.  Demonstrates a one
// level call tracer.

level1();

function level1()
{
   testArgs(1, "ONE", true);
}

function testArgs(a1, a2,a3)
{
   document.write(callerName(arguments));
   document.write("<BR>");
}

function callerName(a1)
{
   myCallerObject = a1.caller.callee;
   myCallerSource = String(myCallerObject);
   mySplitArray1  = myCallerSource.split(" ");
   mySplitArray2  = mySplitArray1[1].split("(");
   myCaller       = mySplitArray2[0];
   return(myCaller);
}
</SCRIPT>

<BODY>
</HTML>
```

See also: `Argument`, Argument list, `Arguments.callee`, `Arguments.caller`, `Arguments.length`, `arguments[]`, `Collection` object, Execution context, Function arguments, Function call, Function call operator ( ), `function( ... )` ..., `Function.arguments[]`, Object inspector, `Object.prototype`, Parameter

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`callee`	1.2 +	5.5 +	4.0 +	5.5 +	-	-	-	DontEnum
`caller`	1.1 +	5.5 +	3.0 +	5.5 +	-	-	-	Warning, DontEnum, Deprecated
`length`	1.1 +	5.5 +	3.0 +	5.5 +	-	-	-	ReadOnly, DontEnum

Cross-references:

ECMA 262 edition 2 – section – 10.1.6

ECMA 262 edition 2 – section – 10.1.8

ECMA 262 edition 2 – section – 15.2.3.1

ECMA 262 edition 3 – section – 10.1.6

ECMA 262 edition 3 – section – 10.1.8

Wrox *Instant JavaScript*, ISBN 1-861001-27-4– page – 27

Arguments.callee (Property)

The `function` object being called.

Availability:	JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0	
Property/method value type:	`Function` object	
JavaScript syntax:	-	*`myArguments`*`.callee`

The value yielded by this property is the `function` object that owns the arguments.

You can work out the calling tree by tracing the callee and caller relationships back up the tree. The callee is a reference to the parent function that owns the `arguments` object.

This has no meaning outside of the context of a function.

See also:	`Arguments` object, `Arguments.caller`, Debugging – client side, `Function` object

Property attributes:

`DontEnum`.

Arguments.caller (Property)

The object that called the function that owns the arguments.

Availability:	JavaScript – 1.1 JScript – 5.5 Internet Explorer – 5.5 Netscape – 3.0 Deprecated	
Property/method value type:	`Arguments.object`	
JavaScript syntax:	-	*myArguments*`.caller`

This property refers to an `Arguments` object belonging to a parent function. Function call tracing can traverse a hierarchy based on `Arguments` objects to unwind a call stack. This might be useful when debugging complex script projects.

You can work out the calling tree by tracing the callee and caller relationships back up the execution context tree. The caller is a reference to the `arguments` object of the caller of the function.

To reference the function that called the current one, use this:

```
arguments.caller.callee
```

To get the name of the function that called the current one use this (so long as the interpreter supports the `name` property on functions):

```
arguments.caller.callee.name
```

With this, you could build a stack trace function that you can call and will unwind the calling context stack to show you how you got to the location you are in. Tools such as this are useful to have around and if they are in a separate `.js` file, you can include them when you need to debug a script problem.

When the `caller` value is Null, it refers to the global code context because there is no `arguments` array in that context – at least not in a web browser. Other host implementations may provide an additional level of arguments according to how the script is executed.

This has no meaning outside of the context of a function.

The example shows how to walk up the calling tree and should yield the following output when it is run:

```
level2

called by level1

called by global level
```

Warnings:

- This property is incorrectly implemented in Netscape 3, which returned a reference to the calling function and not its arguments. Since it works correctly in Netscape 4, you should consider that it is only available there.
- The example shown below did not work correctly on Netscape 6.0 at time of writing.
- This is not part of the ECMA standard and is at some risk of becoming deprecated and removed in later versions. In fact, it is deprecated as of JavaScript version 1.3 and should not be used in new projects.
- It is recommended that you do not build this into functional deployed applications, although the risks involved with using it for debugging are small.

Example code:

```
<HTML>
<BODY>

<SCRIPT>
// A function to extract the calling function name when
// passed the arguments object from a function.  Demonstrates
// how to recursively walk up a call tree.level1();

level1();

function level1()
{
   level2();
}

function level2()
{
   testArgs(1, "ONE", true);
}

function testArgs(a1, a2,a3)
{
   document.write(callerName(arguments));
   document.write("<BR>");
}

function callerName(a1)
{
   if(a1.caller == null)
   {
      return("global level");
   }
   myCallerObject = a1.caller.callee;
   myCallerSource = String(myCallerObject);
   mySplitArray1  = myCallerSource.split(" ");
   mySplitArray2  = mySplitArray1[1].split("(");
   myCaller       = mySplitArray2[0];
   return(myCaller+"<BR> called by "+callerName(a1.caller));
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Arguments` object, `Arguments.callee`, Debugging – client side, `Function` object, `Function.caller`, Hierarchy of objects

Property attributes:

`DontEnum`.

Arguments.length (Property)

The number of arguments passed to a function dictates the length of the array to hold them.

Availability:	JavaScript – 1.1 JScript – 5.5 Internet Explorer – 5.5 Netscape – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myArguments*`.length`

The number of arguments passed to a function when it is called.

The `length` property of the `Arguments` object can be inspected or used in an enumeration loop to access each argument in turn.

Even if no placeholder arguments are specified, you can still call a function and pass as many arguments to it as you like. They will be assembled into an array that you can manipulate in the way you would normally operate on any other array. You can build enumerators to process all the elements and do something with them.

You can compare this value with the `arity` property of the owner `function` object. This will allow you to determine whether the correct number of arguments was passed.

Example code:

```
<SCRIPT>
// Declare a function that processes a variable number of arguments
function summate()
{
   var total = 0;
   for(var ii=0; ii<arguments.length; ii++)
   {
      total += arguments[ii];
   }
   return total;
}

// Call the function
sum = summate(1, 2, 3, 4, 5);

document.write(sum);
</SCRIPT>
```

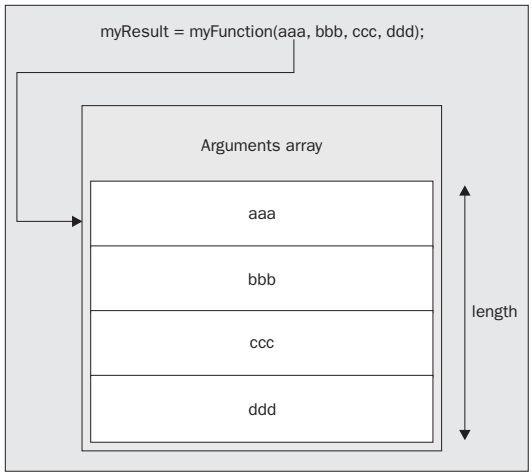
See also:	Argument, Argument list, `Arguments` object, `Collection.length`, `Function.arguments[]`, `Function.arity`, `Function.length`

Property attributes:

`ReadOnly`, `DontEnum`.

Cross-references:

Wrox *Instant JavaScript*, ISBN 1-861001-27-4 – page – 27



arguments[] (Collection)

A property that is available inside a function to access its `Arguments` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 5.5 Internet Explorer – 5.5 Netscape – 3.0

This property is only defined within a function body in a web browser. However, some implementations may provide external arguments via this property.

See also:	`Arguments` object, Function arguments, `Function.arguments[]`

Property attributes:

`ReadOnly`.

argv parameter (Definition)

A command-line argument collection.

Since JavaScript can be used in many environments, it is possible that in a server-side application you will have access to the command-line arguments.

If that is the case, then it is likely that you will have an `argv` property, which contains the argument values.

In general, the first argument is the name of the script or program being executed.

To establish the length of the `argv` array, you can inspect the `argc` value.

The values passed in the `argv` array are likely to be presented as strings, although they may be automatically cast to number, Boolean or other types without you needing to perform any type conversion yourself.

Warnings:

- If you call one script from another, the command-line arguments that were used to invoke the original script may not be propagated unless your calling script makes some arrangements to pass in the arguments it was given. Each script is likely to run in a separate execution context.

See also:	`argc` parameter, Execution context, Execution environment, Host environment, `Host` object, `main()` function

Arithmetic constant (Definition)

A constant derived from arithmetic (numeric) values.

An arithmetic constant is derived from one of the following:

- Unicode character code value of a character constant
- Enumeration constant
- Floating-point constant
- Integer constant
- Math object property
- Number object property
- Global object property

See also:	Constant expression, Floating-point constant, Infinity, Integer constant, `Math.E`, `Math.LN10`, `Math.LN2`, `Math.LOG10E`, `Math.LOG2E`, `Math.PI`, `Math.SQRT1_2`, `Math.SQRT2`, `NaN`, `Number.MAX_VALUE`, `Number.MIN_VALUE`, `Number.NaN`, `Number.NEGATIVE_INFINITY`, `Number.POSITIVE_INFINITY`

Arithmetic operator (Definition)

An operator that works with numeric operands.

The collection of arithmetic operators includes the operators in the following categories:

- Additive operator
- Multiplicative operator
- Postfix operator
- Prefix operator

Warnings:

- Applying some operators causes a strange degenerative effect in the accuracy. On the Macintosh in MSIE 5.0 and in Netscape 4, the following loop generates a very strange sequence of numbers that are quite erroneous:

```
for(myEnum = 1.5; myEnum > -2; myEnum -= 0.1)
{
 document.write(myEnum + "<BR>");
}
```

- There are some very odd and subtle mathematical errors in the arithmetic handling within the Macintosh platform, and it surely must be the platform since the same behavior is found on both MSIE and Netscape.

See also:	Additive operator, Expression, Mathematics, Multiplicative operator, Postfix operator, Prefix decrement (--), Prefix increment (++), Prefix operator, Remainder (%), Remainder then assign (%=), Subtract (-), Subtract then assign (-=), Type conversion

Cross-references:

Wrox Instant JavaScript – page – 18

Arithmetic type (Definition)

A subset of the native types concerned with numeric values.

In the C language, programmers need to be aware of the many and various types of numeric value. JavaScript hides a great deal of this complexity by presenting a Number data type.

However, internally it still uses 32 bit integer values, 16 bit integer values, signed and unsigned integers, and floating-point values.

Arithmetic type values are used with arithmetic operators to build arithmetic expressions.

Characters are maintained as single character strings, but can be represented numerically by converting them to their Unicode code point value using the method `String.charCodeAt()`. You can convert back again using the `String.fromCharCode()` method.

See also:	`String.charCodeAt()`,`String.fromCharCode()`

Array index delimiter ([ ]) (Delimiter)

Access elements of an array with this delimiter.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Depends on array content`	
JavaScript syntax:	-	*myArray*[*anIndex*]
Argument list:	*anIndex*	A legal index value into the array, not greater than the array length

Array elements are indexed by selecting them numerically within the set of elements contained in the array. The `length` property of an array indicates how many indexable locations there are. Array elements begin with the zeroth item.

Storing values into indexes that are higher than the current value of the `length` property will automatically extend the array and reset the `length` property. An array with only one entry in the 100th element (index value 99) is very sparsely populated but still should report a `length` value of 100.

In Netscape, referencing the array with no element delimiters will yield a comma-separated list of the contents of the array. So this:

```
myArray = new Array(6);myArray[0] = 0;myArray[1] = "XXX";myArray[2] = 0;myArray[3]
= "XXX";myArray[4] = 0;myArray[5] = "XXX";document.write(myArray);
```

Yields this when executed:

```
0,XXX,0,XXX,0,XXX
```

Accessing properties of an object by name simply requires the name to be added to the object reference with a dot separator between them. Numeric values cannot be used in this way. You must use a string to name the array element when it is assigned.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

Although JavaScript does not properly support multi-dimensional arrays, you can simulate them by storing references to one array in the elements of another. You need to create a separate array for each row and then one master array to arrange them into a column.

True multi-dimensional arrays would use a notation like this:

```
multiArray[1,2]
```

But in JavaScript we can at least manage this:

```
multiArray[1][2]
```

This is close enough that most programmers will be able to cope with it quite happily.

Another alternative way to do this is to use a single dimensional array, but calculate the indices. For example to make a 5 x 5 array, you would create a single dimensional array that is 25 elements long. Then to reach the rows you use the row number and multiply the value by 5 before adding the column number to access the desired cell. You need to be careful though because if you have an 'off-by-one' error, it all goes wrong.

Warnings:

- Be aware that your script is referring to array elements starting at zero. You can get subtle 'off-by-one' errors if you assume that the array begins at item 1.
- In Netscape 2.02, the `length` property of an array cannot be relied on to hold the right value.
- You should avoid putting spaces into associative names because it introduces a property whose name cannot be reached other than via an array index. Not all implementations will trap this error situation. A property name is an identifier and identifier names cannot contain spaces so it should throw an exception.

Example code:

```
<SCRIPT>
// Multidimensional array simulation
hExtent = 5;
vExtent = 6;
theExtent = hExtent * vExtent;
myArray = new Array(theExtent);
document.write("<TABLE BORDER=1>");
for(vEnum = 0; vEnum < vExtent; vEnum++)
{
   document.write("<TR>");
   for(hEnum = 0; hEnum < hExtent; hEnum++)
   {
      targetCell = (vEnum * hExtent) + hEnum;
      document.write("<TD>");
      document.write(vEnum);
      document.write(",");
      document.write(hEnum);
      document.write(" = ");
      document.write(targetCell);
      document.write("</TD>");
   }
   document.write("</TR>");
}
document.write("</TABLE>");
</SCRIPT>
```

See also:	`Array` object, `Array.length`, Associativity, Multi-dimensional arrays, Off by one errors, Operator Precedence, `Postfix` operator, Property name

Cross-references:

ECMA 262 edition 2 – section – 7.6

ECMA 262 edition 2 – section – 11.2

ECMA 262 edition 3 – section – 7.7

Wrox *Instant JavaScript* ISBN 1-861001-27-4– page – 16

Wrox *Instant JavaScript* ISBN 1-861001-27-4– page – 32

Wrox *Instant JavaScript* ISBN 1-861001-27-4– page – 33

Array literal (Declaration)

A means of creating and initializing an array at once.

Availability:	ECMAScript edition – 3 JavaScript – 1.3 JScript – 5.0 Internet Explorer – 5.0 Netscape – 4.7	
Property/method value type:	`Array` object	
JavaScript syntax:	-	`[ anElement, ... ]`
Argument list:	*`anElement`*	An element to be stored in the array

JavaScript version 1.2 introduces the capability of assigning values to an array as it is created and building the array without first using a constructor.

Now array construction can also be nested to create multi-dimensional arrays.

The result is an array containing the elements defined by the literal expression.

Warnings:

- Netscape 4 does not mind an extra trailing comma (as per the C language convention). To force an undefined element to be assigned to the end of the array, you must place two trailing commas.
- MSIE adds an undefined element for each trailing comma. This means that MSIE creates arrays that are one item longer than Netscape does if there is a trailing comma.
- Some revisions of Netscape exhibit a further problem in that a single numeric value in the square brackets is interpreted as an array length value. This is consistent with the `Array()` constructor but is not correct in this context. You can place a pair of trailing commas there to fix this at the expense of some wasted array items that contain undefined values. This is not a problem on all versions and may be encountered only rarely now.

Example code:

```
<SCRIPT>
// Create a simple array literal
var myArray = [ 100, 1.34, "String text", true, { prop:100 } ];
// Create a nested multi-dimensional array
var matarray = [ [1,0], [0,1]];
// JavaScript expression in arrays
var exprarray = [ Math.random()*10, Math.random()*100 ];
// Sparse array
var sparse = [100, , , , , 1000];
document.write(myArray[2] + "<BR>" + matarray[0,1] + "<BR>" +
   exprarray[1] + "<BR>" + sparse[5]);
</SCRIPT>
```

See also:	`Array` object

Cross-references:

ECMA 262 edition 3 – section – 11.1.4

O'Reilly *JavaScript, The Definitive Guide* ISBN 1-56592-392-8– page – 46

Array object (Object/core)

An object of the class "`Array`".

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0
JavaScript syntax:	- *myArray* = Array - *myArray* = *myVBArray*.toArray() - *myArray* = new Array() - *myArray* = new Array(*aLength*) - *myArray* = new Array(*anItem1*, *anItem2*, *anItem3*, ...)
Argument list:	*aLength* — An optional initial length to set the array to. *anItemN* — A variable number of initial elements to insert into the array.
Object properties:	constructor, index, input, length, prototype
Object methods:	concat(), join(), pop(), push(), reverse(), shift(), slice(), sort(), splice(), toLocaleString(), toSource(), toString(), unshift(), valueOf()

An array is basically an indexed collection of references to other objects or values.

In JavaScript version 1.0, arrays were simple objects and had limited functionality, scarcely enough really to be called arrays. Some commentators argue that the functionality was so limited that they should be flagged as available from version 1.1 of JavaScript only. They were usually simulated by creating an instance of the `Object` object and using its named properties as if the object was an array.

Much additional functionality was added for JavaScript version 1.1. JavaScript version 1.0 lacked the constructors and arrays had no special methods available. The ECMA standard enhances the functionality and Netscape 4 provides additional functionality.

An instance of the class "`Array`" is created by using the `new` operator on the `Array()` constructor. At JavaScript version 1.2, arrays can be created with an Array literal as well. The new object adopts the behavior of the built-in prototype object through the prototype-inheritance mechanisms.

All properties and methods of the prototype are available as if they were part of the instance.

Note that the `index` and `input` properties are available only for arrays that are produced as the result of a `RegExp` match. They are not generally available in Arrays or Collections.

An array is a collection of properties owned by an object and that can be accessed by name or by index position in the array. Because they are collected together and accessible as a set, they may be sorted into the order of the array.

`Array` objects give special treatment to property names, which are numeric values. These are used as an index value and will affect the value of the `length` property. The length supported depends on the platform, but is usually based on a 32 bit integer being used for addressing. That limits the range to 4 Billion array elements.

`Array` objects implement the `Put()` internal function slightly differently from non-array based objects.

The prototype for the `Array` prototype object is the `Object` prototype object.

In the C language, an array is referred to as an aggregate type since it is made from a collection or aggregate of individual members.

Warnings:

- Although arrays were partially supported prior to JavaScript version 1.1, the support was not reliably or completely implemented. There was no way for the script developer to create and modify the arrays. Netscape 2 lacks any realistic array support even though `Array` objects were returned by some object properties.
- The WebTV set top box limits the extent of the `Array` objects to contain only 32,768 elements instead of the 4 Billion or so that is defined as the normal maximum. This is because WebTV uses 16 bit integers for addressing arrays rather than 32 bit integers.

Example code:

```
<SCRIPT>
// Array object demonstration
var weekly_summary = new Array(7);
weekly_summary[1] = 10;
weekly_summary[2] = 25;
var day_names = new Array("Su","Mo","Tu","We","Th","Fr","Sa");
for(var i=0; i<7; i++)
{
   document.write("Summary for day (");
   document.write(day_names[i]);
   document.write(") = ");
   document.write(weekly_summary[i]);
   document.write("<BR>");
}
</SCRIPT>
```

See also:	Aggregate type, Array index delimiter ([ ]), Array literal, `Array()`, `Array()`, `Array.Class`, `Array.length`, `Array.prototype`, `Collection` object, `JavaArray` object, JellyScript, Native object, `Object` object, `String.split()`, `unwatch()`, `VBArray.toArray()`, `watch()`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`constructor`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	2 +	-
`index`	1.2 +	5.5 +	4.0 +	5.5 +	-	-	-	-
`input`	1.2 +	5.5 +	4.0 +	5.5 +	-	-	-	-
`length`	1.0 +	3.0 +	2.0 +	4.0 +	-	-	-	ReadOnly
`prototype`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	2 +	ReadOnly, DontDelete, DontEnum

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`concat()`	1.2 +	3.0 +	4.0 +	4.0 +	-	3.0 +	3 +	Warning
`join()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	-
`pop()`	1.2 +	5.5 +	4.0 +	5.5 +	-	3.0 +	3 +	-
`push()`	1.2 +	5.5 +	4.0 +	5.5 +	-	3.0 +	3 +	-
`reverse()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	-
`shift()`	1.2 +	5.5 +	4.0 +	5.5 +	-	3.0 +	3 +	-
`slice()`	1.2 +	3.0 +	4.0 +	4.0 +	-	3.0 +	3 +	Warning
`sort()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	Warning
`splice()`	1.2 +	5.5 +	4.0 +	5.5 +	-	3.0 +	3 +	Warning
`toLocaleString()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	Warning
`toSource()`	1.3 +	3.0 +	4.06 +	4.0 +	-	-	-	-
`toString()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	Warning
`unshift()`	1.2 +	5.5 +	4.0 +	5.5 +	-	3.0 +	3 +	-
`valueOf()`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	-	-

Cross-references:

ECMA 262 edition 2 – section – 8.6.2.2

ECMA 262 edition 2 – section – 15.4

ECMA 262 edition 3 – section – 8.6.2.2

ECMA 262 edition 3 – section – 15.4

Wrox *Instant JavaScript* ISBN 1-861001-27-4 – page – 15

Array() (Constructor)

An `Array` object constructor.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Array` object	
JavaScript syntax:	- `new Array()`	
	- `new Array(`*`aLength`*`)`	
	- `new Array(`*`anItem1`*`, `*`anItem2`*`, `*`anItem3`*`, ...)`	
Argument list:	*`aLength`*	An optional initial length to set the array to
	`anItemN`	A variable number of initial elements to insert into the array

The `Array()` constructor is used in a `new` expression to manufacture a new instance of the `Array` object.

The arguments passed to the constructor affect the way that the array is initialized.

If no arguments are passed, then an empty array is created. Its length will be zero and it will only have the properties it inherits from its prototype parent.

If it has a single argument, and if that argument is a numeric value that can be realized as an unsigned 32-bit integer with no loss of precision, then it is taken as a length value to initialize the array with. However, according the ECMA standard, a numeric value that is not convertible to a Uint32 should cause a run-time error. This may not be the case with all host implementations and would be considered a minor deviation from the standard. You may find that a single numeric value results in a one-element array containing that value instead of a run-time error. A single argument of non-numeric type results in an array containing one element and having a length value of 1.

If there is more than one argument, then each argument is placed into the array in the order of presentation and the length value set according to the number of arguments provided.

Warnings:

- Netscape 2.02 does not understand the `new Array()` syntax. MSIE 3.02 with JScript 1.0 does not understand `new Array()` either. This can be simulated with objects however.

See also:	`Array` object, Array simulation, `Array()`, `Array.prototype`, `Constructor` function, `constructor` property, `Global` object, `new`, Object constant

Cross-references:

ECMA 262 edition 2 – section – 15.1.3.3

ECMA 262 edition 2 – section – 15.4.1

ECMA 262 edition 2 – section – 15.4.2

ECMA 262 edition 2 – section – 15.4.3.1

ECMA 262 edition 2 – section – 15.4.4

ECMA 262 edition 3 – section – 15.4.2

Wrox *Instant JavaScript* ISBN 1-861001-27-4 – page – 16

Array() (Function)

An `Array` object constructor.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Array` object	
JavaScript syntax:	-	`Array()`
	-	`Array(`*`aLength`*`)`
	-	`Array(`*`anItem1, anItem2, anItem3, ...`*`)`
Argument list:	*`aLength`*	An optional initial length for the array
	`anItemN`	A variable number of initial elements to insert into the array

Calling the `Array()` constructor as a function behaves exactly the same as if it had been called with the `new` operator.

The function call `Array()` is equivalent to the object creation expression `new Array()` with the same arguments. With other primitive objects, calling the constructor as a function carries out a type conversion instead of an object instantiation.

The arguments passed to the constructor affect the way that the array is initialized in the same way as they do with a `new Array()` expression.

See also:	`Array` object, `Array()`, `Array.prototype`, `Cast` operator, `Constructor` function, `constructor` property, Implicit conversion

Cross-references:

ECMA 262 edition 2 – section – 15.1.3.3

ECMA 262 edition 2 – section – 15.4.1

ECMA 262 edition 2 – section – 15.4.2

ECMA 262 edition 3 – section – 15.4.1

Array.Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition – 2

This is an internal property that describes the class that an `Array` object instance is a member of. The reserved words suggest that in the future, this property may be externalized.

See also:	`Array` object, `Class`

Property attributes:

`DontEnum`, `Internal`.

Cross-references:

ECMA 262 edition 2 – section – 8.6.2

ECMA 262 edition 2 – section – 15.4.2.1

ECMA 262 edition 3 – section – 8.6.2

Array.concat() (Method)

Concatenate arrays together.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server – 3.0	
Property/method value type:	`Array object`	
JavaScript syntax:	-	*myArray*.concat(*someValues*, ...)
Argument list:	*someValues*	A sequence of values to concatenate onto the array

The result of this method is a new array consisting of the original array, plus the concatenation.

The values that are passed to the method are added to the end of the array.

If arrays are passed, they are flattened and their individual elements added.

The method returns an array consisting of the original Array plus the concatenated values.

If `Array1` contains "AAA", "BBB", "CCC" and `Array2` contains "000", "111", "222", then the method call `Array1.concat(Array2)` will return an array with all the elements in a single collection. The original arrays will be untouched.

Warnings:

- The `concat()` method will flatten arrays that are passed as arguments. However, it will not recursively flatten multi-dimensional arrays.

Example code:

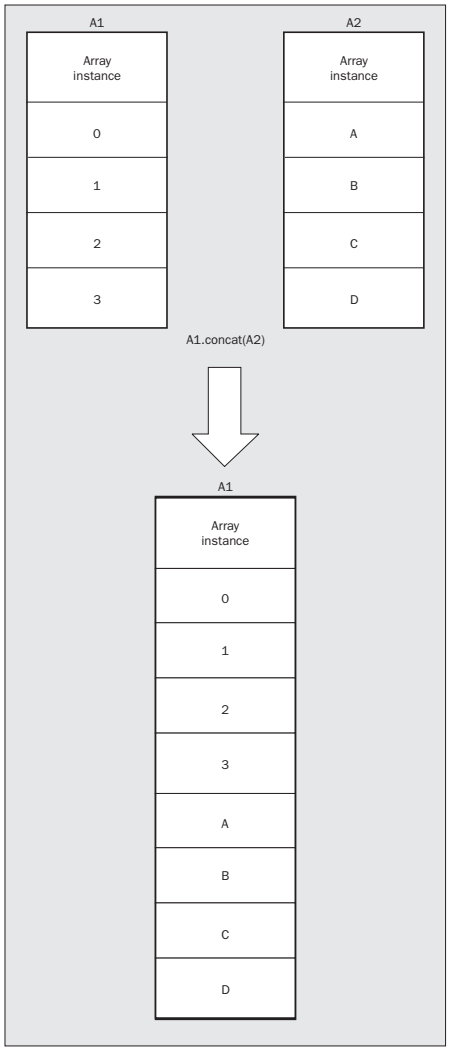
```
<SCRIPT>
// Create two arrays and demonstrate concat() method
myArray1 = new Array("AAA", "BBB", "CCC");
myArray2 = new Array("000", "111", "222");

document.write("Array1<BR>");
displayArrayAsTable(myArray1);
document.write("Array2<BR>")
displayArrayAsTable(myArray2);
document.write("Result returned from Array1.concat(Array2)<BR>")
displayArrayAsTable(myArray1.concat(myArray2));
document.write("Result returned from Array1.concat('AAA')<BR>")
displayArrayAsTable(myArray1.concat("AAA"));
document.write("Result returned from Array1.concat('AAA').concat('DFG')<BR>")
displayArrayAsTable(myArray1.concat("AAA").concat("DFG"));

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
```

```
    for(myIndex = 0; myIndex < myLength; myIndex++)
    {
       document.write("<TR><TD>");
       document.write(myIndex);
       document.write("</TD><TD>");
       document.write(anArray[myIndex]);
       document.write("</TD></TR>");
    }
    document.write("</TABLE><BR><BR>")
}
</SCRIPT>
```

See also:	`Array.prototype,String.concat()`

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.4



Array.constructor (Property)

A reference to a `constructor` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Array constructor`	
JavaScript syntax:	-	*myArray*`.constructor`

The constructor is that of the built-in `Array` prototype object.

You can use this as one way of creating arrays although it is more popular to use the `new Array()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	`Array.length`,`Array.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.4.2

ECMA 262 edition 3 – section – 15.4.2

ECMA 262 edition 3 – section – 15.4.4.1

Array.index (Property)

A special property provided only when the array results from a string match.

Availability:	JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myArray*`.index`

When the `String.match()` method is used, it returns an array as a result. If the match used a pattern that made only a single match (that is, the `g` attribute was not used) then the array returned will have this additional `index` property.

The `index` property will contain the character location within the original string where the match occurred.

See also:	`Array.input`, `RegExp.exec()`, `String.match()`

Array.input (Property)

A special property provided only when the array results from a string match.

Availability:	JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArray*`.input`

When the `String.match()` method is used, it returns an array as a result. If the match used a pattern that made only a single match (i.e. the `g` attribute was not used) then the array returned will have this additional `input` property.

The `input` property will contain a copy of the original string that was searched.

See also:	`Array.index`, `RegExp.exec()`, `String.match()`

Array.join() (Method)

Concatenate array elements to make a string.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArray*`.join(`*aSeparator*`)`
Argument list:	*aSeparator*	A string to place between array elements as the array is concatenated to form a string

The result of this method will be a String primitive containing the array elements interposed with separators.

The elements in the array are converted to strings and are concatenated together to form a larger string. Each element has the separator value placed between it and the next element.

If the separator is not specified, then a single comma is used to join the array elements. This means that if you want no separation between the joined items you should pass an empty string as the separator value.

Example code:

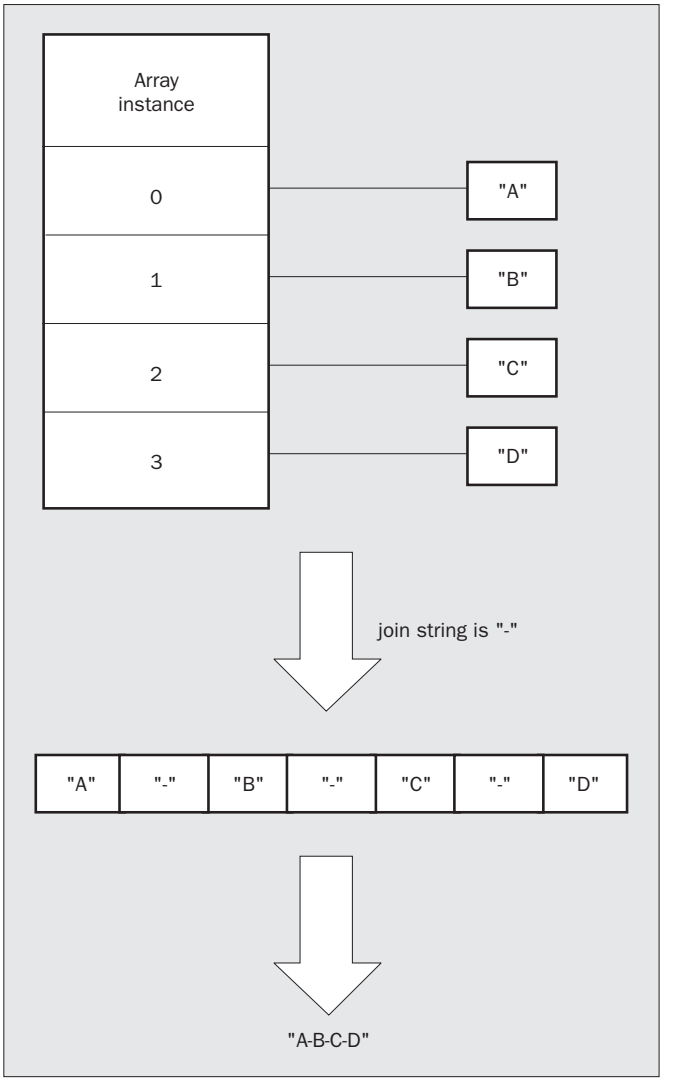
```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Demonstrate array joins
myString1 = "This is a sentence made of words.";
document.write("Original input string<BR>")
document.write(myString1)
myArray = myString1.split(" ");
document.write("<BR><BR>String split into an array<BR>")
displayArrayAsTable(myArray);
myString2 = myArray.join("+");
document.write("<BR><BR>Array joined up as a string<BR>")
document.write(myString2)

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE>")
}
</SCRIPT>
</BODY>
</HTML>
```

See also: `Array.prototype`, `Cast` operator, String concatenate (+), `String.split()`

Cross-references:

ECMA 262 edition 2 – section – 15.4.4.3

ECMA 262 edition 3 – section – 15.4.4.5



Array.length (Property)

The number of elements in an array.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myArray*.length

The `length` property of an array indicates one more than the maximum numeric index value that has currently been used. This is because Array elements start indexing from zero rather than one.

Note that this is not necessarily a count of the exact number of elements in the array just an indication of its range of index values.

Whenever a property is added whose name is an array index, the `length` property is recomputed to allow the length of the array to contain the new element that was added.

Furthermore, when the length value is set explicitly, it may truncate the array and any properties whose names are numeric values that falls outside the bounds indicated by the length value will be deleted.

This does only affect properties that belong to an `Array` object itself though. Any properties that are inherited from a parent or prototype are unaffected.

The maximum value for an Array `length` is 4,294,967,295. This is because a 32 bit integer is used to index the array. Arrays of this length are unlikely to be encountered often! A web page containing an array of that size could take several weeks to download, that is assuming you had 4 GBytes of memory available and that your web browser could address that much storage.

Although this property is marked as `ReadOnly`, there are some sub-classes of the `Array` object that allow you to modify the `length` property directly.

Warnings:

- The `length` property is so unreliable as to be virtually unusable unless you strictly constrain the way you add elements to the array. If you add elements to an array using associative names, the `length` property is not changed at all and will return a zero value.
- If you then add an element to the array whose index is a numeric value, then the `length` property will be set to a value that is one more than the highest numbered numerically indexed item in the array. This following fragment of code yields an array *length* property value of 3:

```
var myArray = new Array();myArray[2] = "ABC";myArray["zero"] =
"ABC";myArray["one"] = "one";myArray["two"] = "two";myArray[0] = "ABC";
```

- This behavior is correct according to the ECMA specification but it is not a genuine measurement of the array length, merely an indication of the highest numerically indexed array element. It should be used for controlling enumeration loops but not for measuring array element item counts.
- WebTV platforms can only address array indices using a 16 bit value and can only access 32,768 items in an array.

See also:	Array index delimiter ([ ]), `Array` object, `Array.constructor`, `Array.prototype`, `Collection.length`, `length`, `NodeList.length`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

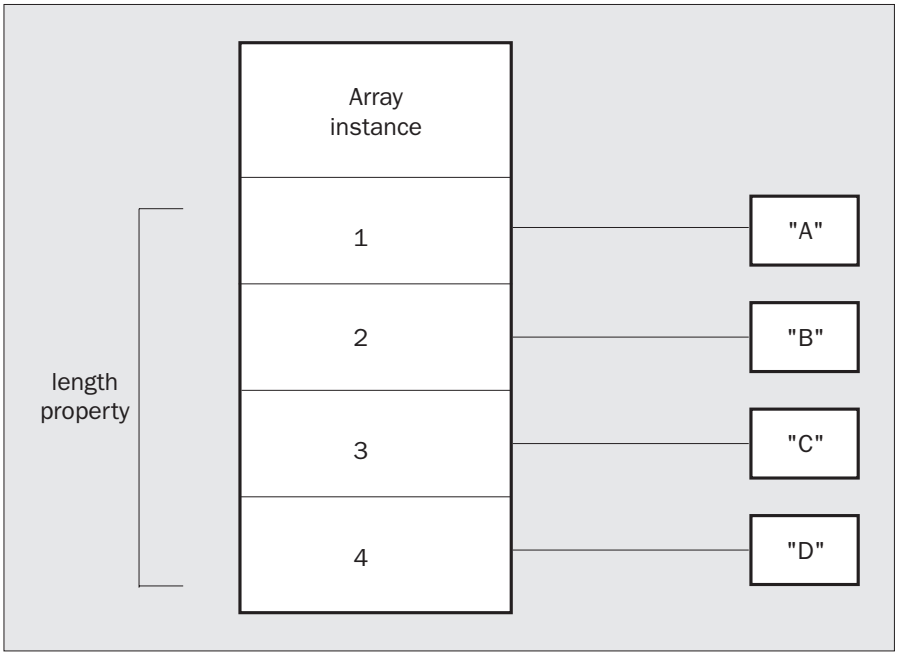
ECMA 262 edition 2 – section – 15.4.2

ECMA 262 edition 2 – section – 15.4.3.2

ECMA 262 edition 2 – section – 15.4.5.2

ECMA 262 edition 3 – section – 15.4.5.2

Wrox *Instant JavaScript* ISBN 1-861001-27-4 – page – 16



Array.pop() (Method)

Pops items off of the end of an array like a FILO stack.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0 Netscape Enterprise Server – 3.0	
Property/method value type:	`Depends on the array content`	
JavaScript syntax:	-	*myArray*`.pop()`

The `pop()` method returns the element at the end of the array. In doing so, it deletes the item from the end of the array reducing the array length by one.

Elements are returned one at a time, even if several were pushed onto the stack together.

Arrays that were pushed onto the stack are returned as arrays.

Although this is very useful for programming stacks, it is not portable enough to deploy in a public facing site.

The result of this method is the item that was on the end of the stack.

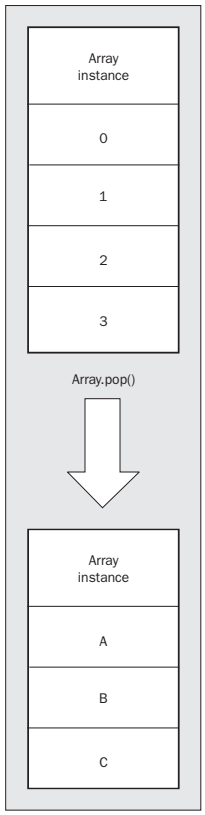
Example code:

```
// Create an array and test the Array.pop() method
myArray = new Array("AAA", "BBB", "CCC");
document.write("Array<BR>")
displayArrayAsTable(myArray);
document.write("Array.pop()<BR>")
document.write(myArray.pop())
document.write("<BR><BR>")
document.write("Array after pop() call<BR>")
displayArrayAsTable(myArray);

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE><BR><BR>")
}
```

See also:	`Array.prototype`, `Array.push()`, Queue manipulation, Stack manipulation

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.6



Array.prototype (Property)

The prototype for the `Array` object, which can be used to extend the interface for all `Array` objects.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Array` object	
JavaScript syntax:	-	`Array.prototype`
	-	*`myArray`*`.constructor.prototype`

The prototype for an array is the original `Array` prototype object.

`Array` objects inherit many methods and properties from their prototype parent. Enquiring the prototype of a an object whose provenance is unknown will tell you what sort of object it is.

`Array` objects inherit these properties from the built-in `Array` prototype:

- `Array.constructor`
- `Array.prototype`

`Array` objects inherit these methods from the built in `Array` prototype:

- `Array.join()`
- `Array.reverse()`
- `Array.sort()`
- `Array.toString()`

Array instances provide these properties themselves even if the prototype has them:

- `Array.length`

The example demonstrates how to provide extensions to all instances of this class by adding a function to the prototype object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that duplicates the item at the end of
// the array
function dupe()
{
   if(this.length != 0)
   {
      myTail = this[this.length-1];
      this[this.length] = myTail;
      return myTail;
   }
}

// Register the new function
Array.prototype.dupe = dupe;

// Create an array and test the Array.dupe() method
myArray = new Array("AAA", "BBB", "CCC");
document.write("Array<BR>");
displayArrayAsTable(myArray);
document.write("Array.dupe()<BR>");
document.write(myArray.dupe());
document.write("<BR>");
document.write("<BR>");
document.write("Array after dupe() call<BR>");
displayArrayAsTable(myArray);
```

```
// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
      document.write("</TD></TR>");
   }
   document.write("</TABLE><BR><BR>")
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Array` object, `Array()`, `Array()`, `Array.concat()`, `Array.constructor`, `Array.join()`, `Array.length`, `Array.pop()`, `Array.push()`, `Array.reverse()`, `Array.shift()`, `Array.slice()`, `Array.sort()`, `Array.splice()`, `Array.toSource()`, `Array.toString()`, `Array.unshift()`, `prototype` property

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 – section – 15.4.3.1

ECMA 262 edition 3 – section – 15.4.3.1

Array.push() (Method)

Pushes items onto the end of an array like a FILO stack.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0 Netscape Enterprise Server – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myArray`*`.push(`*`someValue`*`, ...)`
Argument list:	*`someValue`*	A series of values to be pushed onto the stack

The value is added to the end of the array.

If the value is an array itself, it is not flattened. When it is eventually popped, you get the array back.

If several values are passed to the `push()` method, they will all be added to the stack, but only the last one will be returned.

This modifies the receiving array, increasing the array length by the number of elements that were pushed onto the end.

The result of this method is the new length of the receiving array after the pushed item has been concatenated onto its tail.

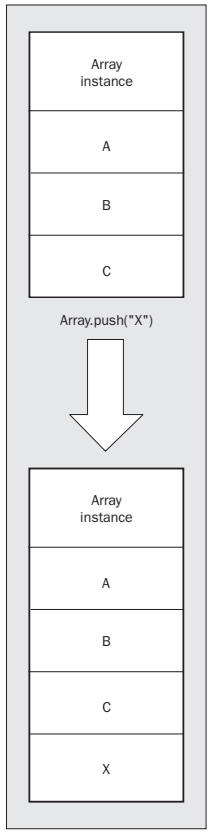
Example code:

```
// Create an array and test the Array.push() method
myArray = new Array("AAA", "BBB", "CCC");
document.write("Array<BR>");
displayArrayAsTable(myArray);
document.write("Array.push()<BR>")
document.write(myArray.push("XXX"));
document.write("<BR><BR>");
document.write("Array after push('XXX') call<BR>");
displayArrayAsTable(myArray);

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE><BR><BR>")
}
```

See also:	`Array.pop()`, `Array.prototype`, `Array.unshift()`, Queue manipulation, Stack manipulation

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.7



Array.reverse() (Method)

Reverse the order of array elements.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Array` object	
JavaScript syntax:	-	*myArray*`.reverse()`

The elements in the array are rearranged into reverse order. The `Array` object is returned as the result.

Note that the `reverse()` method may possibly be applied to other object types. Host objects may support the `reverse()` method, but it will be in an implementation-dependant manner.

The result of this method is the array with its elements in reversed order.

Example code:

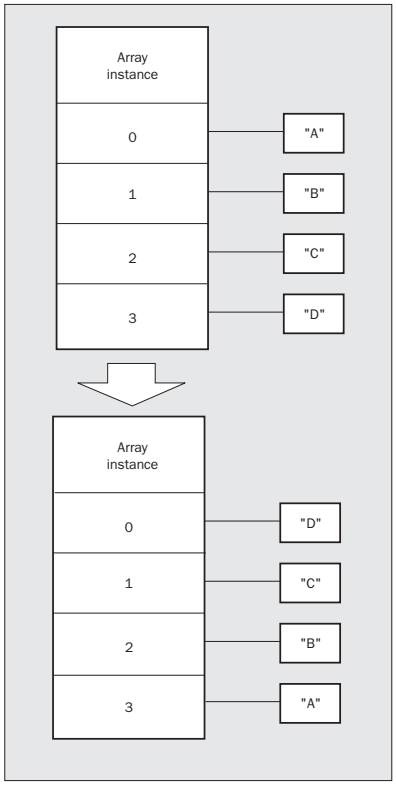
```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Demonstrate array joins
myString1 = "This is a sentence made of words.";
document.write("Original input string<BR>")
document.write(myString1)
myArray = myString1.split(" ");
document.write("<BR><BR>String split into an array<BR>")
displayArrayAsTable(myArray);
myArray.reverse();
document.write("<BR><BR>Array reversed<BR>")
displayArrayAsTable(myArray);
myString2 = myArray.join(" ");
document.write("<BR><BR>Array joined up as a string<BR>")
document.write(myString2)

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE>")
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Array.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.4.4.4

ECMA 262 edition 3 – section – 15.4.4.8



Array.shift() (Method)

Pull off of a stack whose access is FILO from the start rather than the end.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0 Netscape Enterprise Server – 3.0	
Property/method value type:	Depends on array content	
JavaScript syntax:	-	*myArray*.shift()

This method pulls an item from the front of the array and removes that item.

The array elements are all moved down one index position.

This modifies the array in place.

The result of this method is the item that is deleted from the front of the stack.

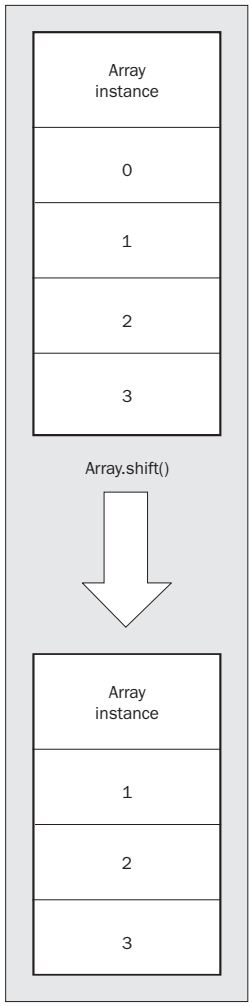
Example code:

```
// Create an array and test the Array.shift() method
myArray = new Array("AAA", "BBB", "CCC");
document.write("Array<BR>")
displayArrayAsTable(myArray);
document.write("Array.shift()<BR>")
document.write(myArray.shift())
document.write("<BR>")
document.write("<BR>")
document.write("Array after shift() call<BR>")
displayArrayAsTable(myArray);

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE><BR><BR>")
}
```

See also: `Array.prototype`, `Array.unshift()`, Queue manipulation, Stack manipulation

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.9



Array.slice() (Method)

Slice out a sub-array from the receiving array.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server – 3.0	
Property/method value type:	`Array` object	
JavaScript syntax:	-	*myArray*`.slice(`*aRange*`)`
Argument list:	*aRange*	A range of array elements

This method returns the sliced-out sub-array presented as a new array.

The range values indicate which part of the receiving array is to be sliced out.

A positive value in the range specifier indicates a particular cell.The first cell index is 0.

A negative value in the range specifier indicates a cell counted back from the end of the array. The last cell is index -1.

If only one value is indicated in the range specifier, then the second is assumed to be the end of the array.

The first specifier should indicate an element earlier than the second although some implementations may check and swap as necessary.

Warnings:

- There are some bugs in the way this works in MSIE. These are mostly to do with specifying negative values for range specifiers. These bugs are still extant as of version 5 of MSIE so you should avoid using the negative indices, and instead measure the length of the array and compute a positive index to use instead. Be careful of 'off-by-one' errors when you do this.

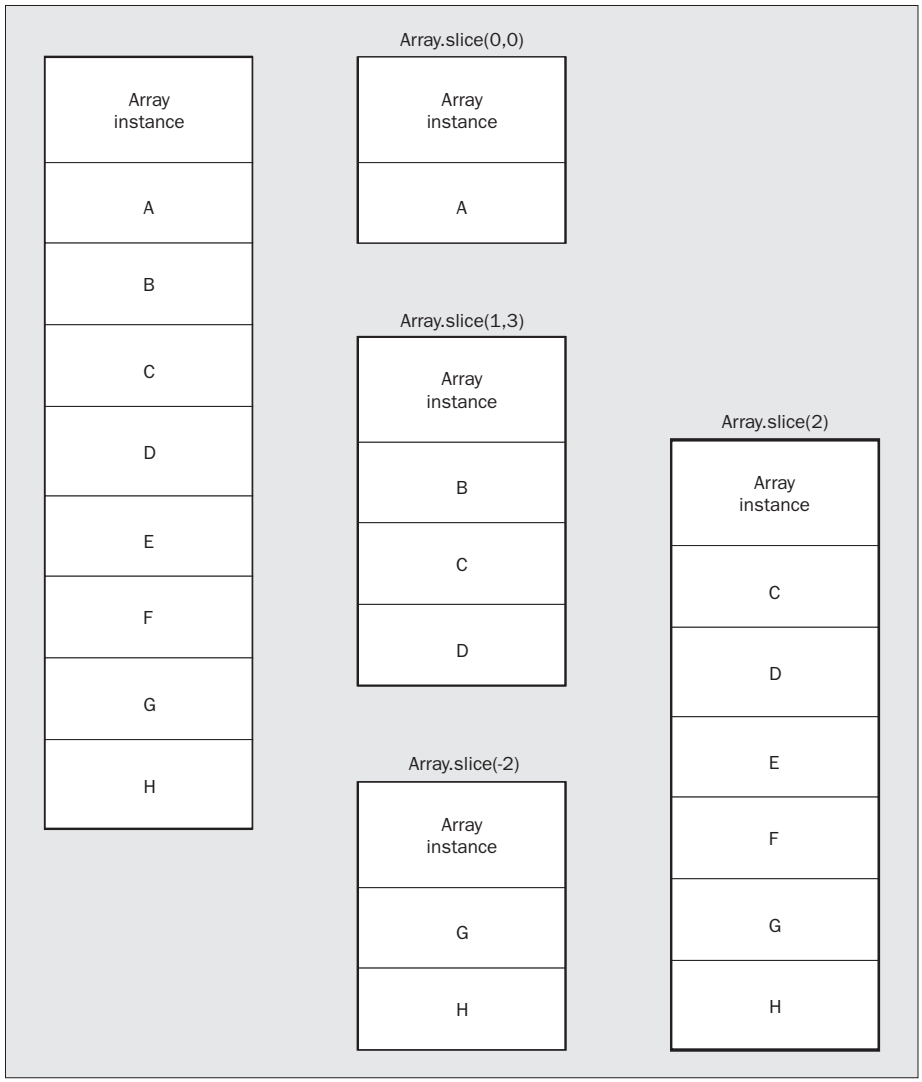
Example code:

```
// Create an array and test the Array.slice() method
myArray = new Array("AAA", "BBB", "CCC", "DDD", "EEE");
document.write("Array<BR>")
displayArrayAsTable(myArray);
document.write("Array.slice(3)<BR>")
displayArrayAsTable(myArray.slice(3))
document.write("Array.slice(2,4)<BR>")
displayArrayAsTable(myArray.slice(2,4))
document.write("Array.slice(-1)<BR>")
displayArrayAsTable(myArray.slice(-1))
document.write("Array.slice(-3)<BR>")
displayArrayAsTable(myArray.slice(-3))

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE><BR><BR>")
}
```

See also:	`Array.prototype`, Off-by-one errors

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.10



Array.sort() (Method)

Sort the elements in an array.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0

Property/method value type:	`Array` object	
JavaScript syntax:	-	*myArray*`.sort()`
	-	*myArray*`.sort(`*aComparator*`)`
Argument list:	*aComparator*	A function object that will compare two items and returns a flag indicating their order in the sort collating sequence

The elements in the array are sorted in place and the sorted array is returned as a result of this method. The argument provides a comparator function to determine the relationship between any two items.

The comparator function is necessary if you want to sort into any order other than alphabetically ascending. You can observe the operation of the comparator by placing `document.write()` methods into its source text. These will demonstrate how the comparison is called during the sort.

In the example, a comparator function shows how to custom-sort items. The example demonstrates sorting by length rather than charset collation sequence. You must make sure the comparator returns one of the following three values:

- Negative integer – signifies that the first argument is less than the second.
- Zero – Signifies that both arguments are the same.
- Positive integer – Signifies that the first argument is larger than the second.

You can reverse the sort direction by negating the result returned by this comparator function.

In the example, a more highly optimized comparator is shown as well. The more lengthy version is presented first to illustrate the algorithmic requirements of the comparator, but the second is functionally identical and can be accomplished in one line and therefore the sort is much faster.

The result of this method is the array with its elements sorted according to the comparator.

Warnings:

- According to the ECMA standard, this sort may not be stable.
- The `sort()` method is generic and may be applied to non-array objects. However, some objects may not be conducive to sorting like this and the exact behavior may be host implementation-dependant in some cases.
- The custom comparator is not supported by the WebTV platform as of the Summer 2000 release of the JellyScript interpreter.
- It is easy to make the mistake of returning `true` and `false` as a result of comparing the two values. For example the following is wrong:

```
function compare(aValue1, aValue2){
   if(aValue1.length <= aValue2.length)
   {
      return false;
   }
   return true;}
```

- This will not work properly and the resulting sort will be incorrect.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Demonstrate array joins
myString1 = "This is a sentence made of words.";
document.write("Original input string<BR>")
document.write(myString1)
myArray = myString1.split(" ");
document.write("<BR><BR>String split into an array<BR>")
displayArrayAsTable(myArray);
myArray.sort(compare);
document.write("<BR><BR>Array sorted<BR>")
displayArrayAsTable(myArray);
myString2 = myArray.join(" ");
document.write("<BR><BR>Array joined up as a string<BR>")
document.write(myString2)

// Comparator function
function compare(aValue1, aValue2)
{
   if(aValue1.length < aValue2.length)
   {
      return -1;
   }

   if(aValue1.length > aValue2.length)
   {
      return 1;
   }

   return 0;
}

// Optimised comparator function
function optimalCompare(aValue1, aValue2)
{
   return (aValue1.length - aValue2.length);
}

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD><TD>");
```

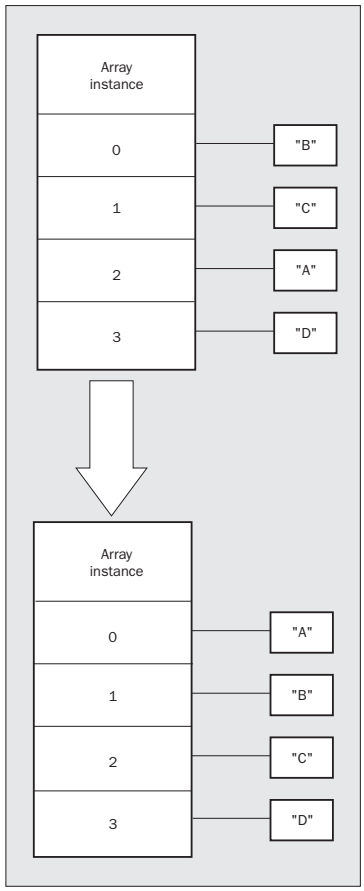
```
        document.write(anArray[myIndex].length);
        document.write("</TD></TR>");
    }
    document.write("</TABLE>")
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Array.prototype`, JellyScript

Cross-references:

ECMA 262 edition 2 – section – 15.4.4.5

ECMA 262 edition 3 – section – 15.4.4.11



Array.splice() (Method)

An array editing tool.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0 Netscape Enterprise Server – 3.0	
Property/method value type:	`Array` object	
JavaScript syntax:	-	*myArray*`.splice(`*startPos*`, `*aCount*`, `*newElements*`)`
Argument list:	*aCount*	An optional count of items to remove
	newElements	An optional list of items to add
	startPos	An entry at which to start splicing

The start position indicates where the splicing is to occur. If there are no other arguments, then the remainder of the array is truncated.

If there is a count argument present, only that number of items will be removed and the subsequent ones shuffled up to be adjacent to the front section of the array.

Any additional arguments are taken to be values to be inserted. They are not evaluated according to the techniques used by the `Array.concat()` method. If arrays are specified, they will not be flattened but will be inserted as they are.

This method operates on the array in place, therefore it modifies the original receiving array.

Specifying a count value of zero provides the functionality of an `insert()` method.

Specifying a count value larger than the array length does not cause an error, but instead truncates the array, behaving like a `replace()` method.

The method call returns an array containing the elements that were removed. This provides an alternative to the `slice()` method, but you should use `slice()` for portability since some MSIE browsers do not support the `splice()` method.

Warnings:

- In Netscape 4, there are some bugs with the values that get returned by this method. The receiving array does get spliced, but the deleted items are not always properly returned.

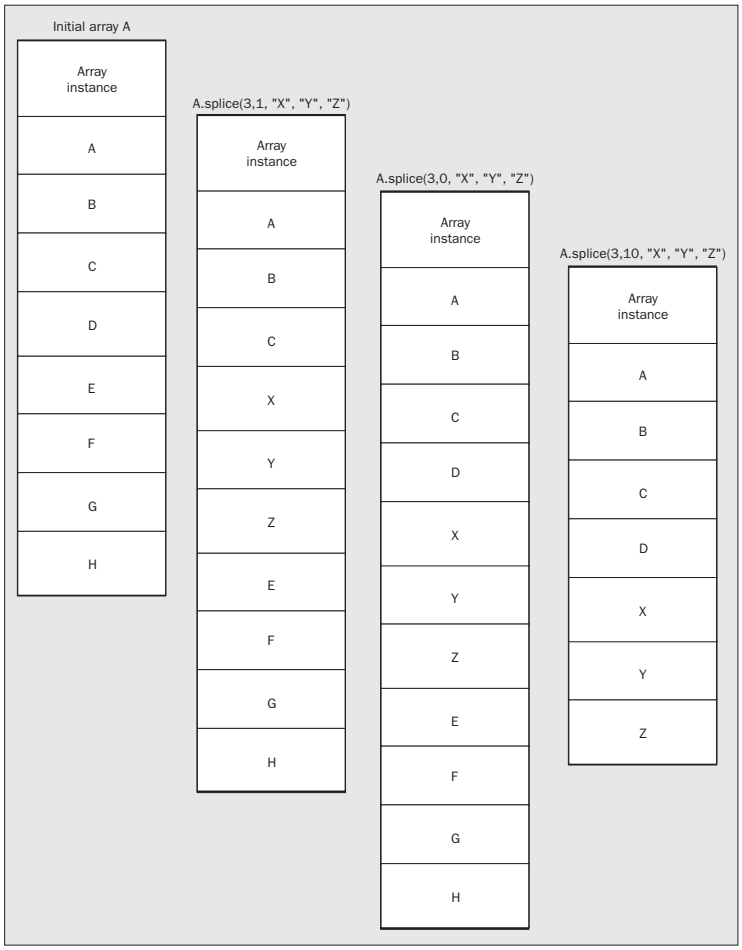
Example code:

```
// Create an array and test the Array.splice() method
myArray = new Array("AAA", "BBB", "CCC", "DDD", "EEE");
document.write("Array<BR>")
displayArrayAsTable(myArray);
document.write("Array.splice() result<BR>")
displayArrayAsTable(myArray.splice(3, 1, "XXX", "YYY", "ZZZ"));
document.write("Array after splice<BR>")
displayArrayAsTable(myArray);

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE><BR><BR>")
}
```

See also:	`Array.prototype`

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.12



Array.toLocaleString() (Method)

Returns a string primitive version of the array taking the present locale into account during the translation.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myArray`*`.toLocaleString()`

The locale context supplies some special conversion rules for strings. Depending on the locale, this might include special characters or a means of using double-byte characters. It may also affect the direction of the text, for certain Asian locales for example.

Warnings:

- The ECMA standard reserves the first argument of this method for future use. It does not specify what that is, but warns against implementations extending the syntax to include its use.

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.3

Array.toSource() (Method)

Output an array formatted as an Array literal contained in a string.

Availability:	JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArray*`.toString()`

This is an alternative way to deliver a string version of an array. In this case, it is formatted as an Array literal and can then be used in an `eval()` function to assign another array. It means that Arrays can be deep copied more easily.

This functionality was previously available in Netscape 4 when the `toString()` method was executed in a `<SCRIPT>` block that was evaluated under explicit JavaScript version 1.2 language selection.

If you run the example, it should yield something like this:

```
["one", 2, "III"]
```

This is quite different from the result of a `toString()` method which would yield this for the same array contents:

```
one,2,III
```

The result of this method is a String primitive version of the array formatted as an Array literal.

Example code:

```
// Create an array and display its source
myObject = new Array("one", 2, "III");
document.write(myObject.toSource());
```

See also:	`Array.prototype,Array.toString()`

Array.toString() (Method)

Return a string primitive version of an object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myArray*`.toString()`

The elements in the array are converted to strings and are concatenated together to form a larger string.

This is functionally identical to using the `join()` method with no join string argument.

If you run the example, it will yield the following:

```
one,2,III
```

This is quite different from what you get if you use the `toSource()` method, which presents this result:

```
["one", 2, "III"]
```

The result of this method is a String primitive version of the array assembled by concatenation.

Warnings:

- Netscape supports a special conversion mechanism if this method is invoked within a `<SCRIPT>` HTML tag whose `LANGUAGE` attribute is set to the "JavaScript1.2" value.
- In that circumstance, the array is presented with enclosing square brackets. This means that it can be used as an array literal in an `eval()` function. This behavior was added in anticipation of the ECMA specification supporting some additional functionality. However, the standard mandates very specific behavior for `toString()`.
- In JavaScript 1.3, the `toString()` behavior will revert to what was expected. Because this source form output is so useful, it will continue to be supported by a new method called `toSource()`.

Example code:

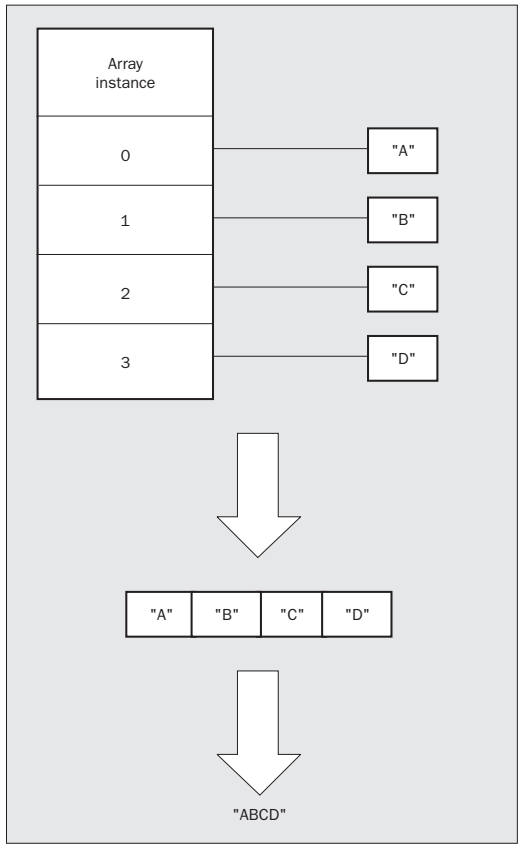
```
// Create an array and display it as a string
myObject = new Array("one", 2, "III");
document.write(myObject.toString());
```

See also:	`Array.prototype`, `Array.toSource()`, `Cast` operator, String concatenate (+), `toString()`

Cross-references:

ECMA 262 edition 2 – section – 15.4.4.2

ECMA 262 edition 3 – section – 15.4.4.2



Array.unshift() (Method)

Push onto a stack whose access is FILO from the start rather than the end.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0 Netscape Enterprise Server – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myArray`*`.unshift(`*`someValue, ...`*`)`
Argument list:	*`someValue`*	A series of values to be pushed onto the stack

This operates very like the `Array.push()` method except that items are added to the front of the stack rather than the end of the stack. The items are also pushed in reverse order if several are presented at once. That is to say, the order of presentation is preserved within the array.

When the push is completed, the item at the front of the array is returned.

The number of items that were added increases the array length.

If arrays are presented, they will be pushed on as they are and not flattened. When they are subsequently removed from the stack, they will still be arrays.

This method modifies the array in place.

The result of this method is the new length of the receiving array after the pushed item has been concatenated onto its front.

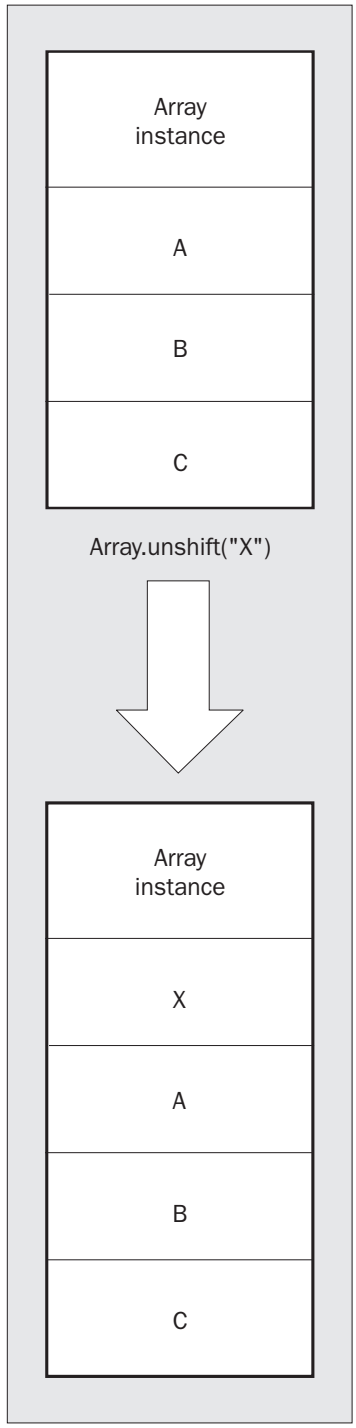
Example code:

```
// Create an array and test the Array.unshift() method
myArray = new Array("AAA", "BBB", "CCC");
document.write("Array<BR>")
displayArrayAsTable(myArray);
document.write("Array.unshift()<BR>")
document.write(myArray.unshift("XXX"))
document.write("<BR><BR>")
document.write("Array after unshift('XXX') call<BR>")
displayArrayAsTable(myArray);

// Display an array in a table
function displayArrayAsTable(anArray)
{
   myLength = anArray.length;
   document.write("<TABLE BORDER=1>");
   for(myIndex = 0; myIndex < myLength; myIndex++)
   {
      document.write("<TR><TD>");
      document.write(myIndex);
      document.write("</TD><TD>");
      document.write(anArray[myIndex]);
      document.write("</TD></TR>");
   }
   document.write("</TABLE><BR><BR>")
}
```

See also: `Array.prototype`, `Array.push()`, `Array.shift()`, Queue manipulation, Stack manipulation

Cross-references:

ECMA 262 edition 3 – section – 15.4.4.13


Array
instance
A
B
C
Array.unshift("X")
Array
instance
X
A
B
C

Array.valueOf() (Method)

Returns the contents of the array converted to a native primitive value.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myObject*`.valueOf()`

The primitive value of the receiving object is returned by this method. Because an array is an aggregation of many elements, a simple type conversion is not appropriate. The individual elements are converted to string values and are then concatenated together and returned as a single string primitive value. This applies even if an array comprises a collection of numeric values.

See also:	`valueOf()`

Array simulation (Definition)

A means of simulating arrays in JavaScript.

With a constructor, you can simulate arrays by making them from objects and property components. This may be useful if you want to run an array-based script in a very old JavaScript implementation although these days that likelihood is diminishing rapidly.

This was necessary in JavaScript version 1.0. Numbered index locations within an `Object` object could simulate `Array` objects. Named items simply allocate the next available numbered entry.

Thankfully we don't have to do this anymore.

Warnings:

- In Netscape 2.02 and MSIE 3.02 you can operate on existing arrays, but you cannot make a new one.
- In Netscape 2.02 the array length value does not work properly.

Example code:

```
// Simulate an array with an Object object
myArray = new Object();
myArray[0] = "One";
myArray[1] = "Two";

// Simulate an array with a constructor
function SimArray(aSize)
{
```

```
    this.length = aSize;
    for(var index = 0; index<aSize; index++)
    {
       this[index] = 0;
    }
    return this;
}

// Now make a simulated array
myArray = new SimArray(12);
```

See also:	`Array()`, `Cast` operator, `Constructor` function

Cross-references:

Wrox *Instant JavaScript* ISBN 1-861001-27-4 – page – 32

ASCII (Standard)

A table of seven-bit binary numbers that encode the alphabet and other symbols.

ASCII stands for American Standard Code for Information Interchange. It describes an encoding for letters, numbers and punctuation symbols that can be realized in seven bits. It uses only seven of the 8 bits for historical reasons to allow the eighth bit to be used for parity control when the characters are transmitted through serial interfaces.

Many of the character codes are reserved to send control signals to terminals and to manage the communications. Modern networking provides this capability outside of the character encoding.

There is an extended ASCII encoding that provides all 8 bits for character code mapping. This defines the upper 128 characters in addition to the lower 128 characters in the 7-bit representation.

There are many alternative interpretations of the ASCII character set that allow for national extensions to the character set. In some cases, this may only result in the replacement of a few currency symbols.

JavaScript uses the Unicode character set. The lower 128 characters of Unicode are purposely mapped to the ASCII character. ASCII is described here to provide help when you are exchanging data files with ASCII-based systems or applications.

It may also be useful in some situations if you are using JavaScript to drive a serial interface to control some external system. Whether you could do that would depend on the hosting environment. A browser wouldn't give you those capabilities, but an embedded JavaScript interpreter in a process control system may well allow you to do that sort of thing.

This table summarizes the lower 128 characters in the ASCII character set:

Dec	Hex	Sym	Unicode	Description
000	00	NUL	\u0000	<ctrl-@> Null character
001	01	SOH	\u0001	<ctrl-A> Start of header
002	02	STX	\u0002	<ctrl-B> Start of text

Table continued on following page

Dec	Hex	Sym	Unicode	Description
003	03	ETX	\u0003	<ctrl-C> End of text
004	04	EOT	\u0004	<ctrl-D> End of transmission
005	05	ENQ	\u0005	<ctrl-E> Enquiry
006	06	ACK	\u0006	<ctrl-F> Positive acknowledge
007	07	BEL	\u0007	<ctrl-G> Alert (bell)
008	08	BS	\u0008	<ctrl-H> Backspace
009	09	HT	\u0009	<ctrl-I> Horizontal tab
010	0A	LF	\u000A	<ctrl-J> Line feed
011	0B	VT	\u000B	<ctrl-K> Vertical tab
012	0C	FF	\u000C	<ctrl-L> Form feed
013	0D	CR	\u000D	<ctrl-M> Carriage return
014	0E	SO	\u000E	<ctrl-N> Shift out
015	0F	SI	\u000F	<ctrl-O> Shift in
016	10	DLE	\u0010	<ctrl-P> Data link escape
017	11	DC1	\u0011	<ctrl-Q> Device control 1 (XON)
018	12	DC2	\u0012	<ctrl-R> Device control 2 (tape on)
019	13	DC3	\u0013	<ctrl-S> Device control 3 (XOFF)
020	14	DC4	\u0014	<ctrl-T> Device control 4 (tape off)
021	15	NAK	\u0015	<ctrl-U> Negative acknowledgement
022	16	SYN	\u0016	<ctrl-V> Synchronous idle
023	17	ETB	\u0017	<ctrl-W> End of transmission block
024	18	CAN	\u0018	<ctrl-X> Cancel
025	19	EM	\u0019	<ctrl-Y> End of medium
026	1A	SUB	\u001A	<ctrl-Z> Substitute
027	1B	ESC	\u001B	<ctrl-[> Escape
028	1C	FS	\u001C	<ctrl-\> File separator (Form separator)
029	1D	GS	\u001D	<ctrl-]> Group separator
030	1E	RS	\u001E	<ctrl-^> Record separator
031	1F	US	\u001F	<ctrl-_> Unit separator
032	20	SP	\u0020	Space
033	21	!	\u0021	Exclamation point (bang)
034	22	"	\u0022	Double quote
035	23	#	\u0023	Hash (number sign, pound sign, sharp)
036	24	$	\u0024	Dollar sign (buck)
037	25	%	\u0025	Percent sign
038	26	&	\u0026	Ampersand
039	27	'	\u0027	Apostrophe (single quote)
040	28	(	\u0028	Left parenthesis

Table continued on following page

Dec	Hex	Sym	Unicode	Description
041	29	)	\u0029	Right parenthesis
042	2A	*	\u002A	Asterisk (star)
043	2B	+	\u002B	Plus sign
044	2C	,	\u002C	Comma
045	2D	-	\u002D	Minus sign (hyphen)
046	2E	.	\u002E	Period (full stop, dot, point)
047	2F	/	\u002F	Slash (virgule, solidus)
048	30	0	\u0030	-
049	31	1	\u0031	-
050	32	2	\u0032	-
051	33	3	\u0033	-
052	34	4	\u0034	-
053	35	5	\u0035	-
054	36	6	\u0036	-
055	37	7	\u0037	-
056	38	8	\u0038	-
057	39	9	\u0039	-
058	3A	:	\u003A	Colon
059	3B	;	\u003B	Semi-colon
060	3C	<	\u003C	Left caret (less than, left angle bracket)
061	3D	=	\u003D	Equal sign
062	3E	>	\u003E	Right caret (greater than, right angle bracket)
063	3F	?	\u003F	Question mark
064	40	@	\u0040	Commercial at sign
065	41	A	\u0041	-
066	42	B	\u0042	-
067	43	C	\u0043	-
068	44	D	\u0044	-
069	45	E	\u0045	-
070	46	F	\u0046	-
071	47	G	\u0047	-
072	48	H	\u0048	-
073	49	I	\u0049	-
074	4A	J	\u004A	-
075	4B	K	\u004B	-
076	4C	L	\u004C	-
077	4D	M	\u004D	-
078	4E	N	\u004E	-

Table continued on following page

Dec	Hex	Sym	Unicode	Description
079	4F	O	\u004F	-
080	50	P	\u0050	-
081	51	Q	\u0051	-
082	52	R	\u0052	-
083	53	S	\u0053	-
084	54	T	\u0054	-
085	55	U	\u0055	-
086	56	V	\u0056	-
087	57	W	\u0057	-
088	58	X	\u0058	-
089	59	Y	\u0059	-
090	5A	Z	\u005A	-
091	5B	[	\u005B	Left square bracket
092	5C	\	\u005C	Backslash (reverse solidus)
093	5D	]	\u005D	Right square bracket
094	5E	^	\u005E	Circumflex accent
095	5F	_	\u005F	Underscore (low line)
096	60	`	\u0060	Grave accent (back quote, back tick)
097	61	a	\u0061	-
098	62	b	\u0062	-
099	63	c	\u0063	-
100	64	d	\u0064	-
101	65	e	\u0065	-
102	66	f	\u0066	-
103	67	g	\u0067	-
104	68	h	\u0068	-
105	69	i	\u0069	-
106	6A	j	\u006A	-
107	6B	k	\u006B	-
108	6C	l	\u006C	-
109	6D	m	\u006D	-
110	6E	n	\u006E	-
111	6F	o	\u006F	-
112	70	p	\u0070	-
113	71	q	\u0071	-
114	72	r	\u0072	-
115	73	s	\u0073	-
116	74	t	\u0074	-

Table continued on following page

Dec	Hex	Sym	Unicode	Description
117	75	u	\u0075	-
118	76	v	\u0076	-
119	77	w	\u0077	-
120	78	x	\u0078	-
121	79	y	\u0079	-
122	7A	z	\u007A	-
123	7B	{	\u007B	Left brace (left curly bracket)
124	7C	|	\u007C	Verical line (bar, pipe)
125	7D	}	\u007D	Right brace (right curly bracket)
126	7E	~	\u007E	Tilde
127	7F	DEL	\u007F	Delete

See also:	Character set, Character-case mapping, Control character, Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), `isLower()`, `isUpper()`, Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==)

ASP (Object model)

The object model inside an ASP server module.

As of the time of writing the ASP object model is at version 3.0 and is now shipped with Windows 2000 as part of the core OS. It is a mechanism that enhances the Microsoft IIS product to provide server-side dynamically generated pages and uses JScript 5.0 as its programming language. It also supports VBScript.

Code that is executed in an ASP page is delimited with a special tag pair that does not conform to the HTML standards, but nevertheless should be ignored by browsers if the unprocessed pages ever escape out of the server.

Here is an ASP tag pair with an example fragment of code:

```
<%Response.Write('<HR>');%>
```

More detailed and in-depth information on ASP can be found in the Wrox *ASP 3.0 Programmer's Reference* ISBN 1-861003-23-4.

ASP (Product)

Active Server Pages. A Microsoft product.

See also:	ADO, Active Server Pages

Assign value (=) (Operator/assignment)

Assign one operand to a left value.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Depends on right value`	
JavaScript syntax:	-	*`anLValue = anExpression`*
Argument list:	*`anExpression`*	Some operation that yields a suitable value to assign
	`anLValue`	A target that can be assigned to

The expression value on the right is assigned to the target operand on the left.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

The source expression to the right is called an RValue, the target expression to the left is called an LValue. The LValue must be capable of having something assigned to it and the RVlaue must evaluate to a meaningful and compatible value or a run-time exception will be thrown.

Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.
- Be careful not to confuse the single equals with the double equals. Placing a double equals in place of an assignment will do a comparison without assigning the result. This is less dangerous than mistakenly assigning a value where you intended to compare for equality. The interpreter may be forgiving enough that a run-time error isn't generated, but the side effects could be subtle and make it hard to diagnose the cause.

See also:	= (Assign), Add then assign (+=), Associativity, Concatenate then assign (+=), Equal to (==), `Location.assign()`, LValue, Multiply then assign (*=), Operator Precedence, Reference, Remainder then assign (%=), Subtract (-), `var`

Cross-references:

ECMA 262 edition 2 – section – 10.1.3

ECMA 262 edition 2 – section – 11.1.2

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 3 – section – 10.1.3

ECMA 262 edition 3 – section – 11.1.2

ECMA 262 edition 3 – section – 11.13

ECMA 262 edition 3 – section – 12.2

Assignment expression (Definition)

An expression that causes an assignment as a by-product.

Availability:	ECMAScript edition – 2

Assignment expressions can be broken down into a two-operand expression with the result being assigned to the value on the left.

Note that assignment expressions can be used in their entirety as an RValue. You may want to `void` the assignment expression in some cases to prevent the result of the assignment being used inadvertently as an `HREF`.

See also:	Add then assign (+=), Assignment operator, Concatenate then assign (+=), Expression, LValue, Multiply then assign (*=), Remainder then assign (%=), RValue, Subtract (-), `var`, Variable statement, `void`

Cross-references:

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 3 – section – 12.2

Assignment operator (Definition)

An operator that causes an assignment as a by-product.

Availability:	ECMAScript edition – 2

Here is a table summarizing the assignment operators, most of which can be secondarily classified as members of other operator categories:

Operator:	Equivalent:	Meaning:
=	a = b	Simple assignment to an LValue
+=	a = a + b	Add and assign to an LValue
-=	a = a – b	Subtract and assign to an LValue
*=	a = a * b	Multiply and assign to an LValue
/=	a = a / b	Divide and assign to an LValue
%=	a = a % b	Remainder and assign to an LValue
&=	a = a & b	Bitwise AND and assign to an LValue
|=	a = a | b	Bitwise inclusive OR and assign to an LValue
^=	a = a ^ b	Bitwise exclusive XOR and assign to an LValue
<<=	a = a << b	Bitwise shift left and assign to an LValue
>>=	a = a >> b	Bitwise shift right and assign to an LValue
>>>=	a = a >>> b	Bitwise shift right (unsigned) and assign to an LValue
++	a = a + 1	Increment LValue
--	a = 1 – 1	Decrement LValue

Assignment operators include the simple assignment as well as the compound OP= form where OP is one of the shift, bitwise, multiplicative, or additive operators.

Warnings:

- Note that these operators are destructive. That is one of the source operands is overwritten by the result. In most implementations, it is unlikely that this compound assignment executes more efficiently than the long form version, which reads more clearly and is less prone to accidental damage.
- Although the ECMA standard describes the algorithms to be used for evaluating operators, there is no guarantee that the operands themselves will be evaluated in any particular order. The right one might appear to be the sensible choice for being evaluated first, since the final value of the left one is dependant on it. However, this is implementation-dependant and certain interpreter designs are based around a recursive descent model, which may partially evaluate the left operand before pausing momentarily while the right is evaluated.
- The operand on the left of the operator must be a modifiable LValue. You cannot use these compound operators with a pair of constant literal values although the right-hand operand can be a constant. The left one must be capable of being assigned to.
- It is a general assumption that the left value will be a single variable. However it could be an array element or object property in which case the resolution of the identifier may cause some side effects that are undesirable and may interact with the right-hand operand value.
- The compound operators are considered to be a single token and the characters that compose them may not be separated by whitespace. The operator should be separated from the operands by whitespace however. Some implementations may forgive the lack of whitespace, but this could lead to ambiguities during interpretation. Such errors may be difficult to diagnose.

See also:	= (Assign), Add then assign (+=), Assignment expression, Bitwise AND then assign (&=), Bitwise OR then assign (|=), Bitwise shift left then assign (<<=), Bitwise shift right and assign (>>=), Bitwise unsigned shift right and assign (>>>=), Bitwise XOR and assign (^=), Concatenate then assign (+=), Divide then assign (/=), Multiply then assign (*=), Operator, Postfix expression, Prefix expression, Remainder then assign (%=), Subtract then assign (-=), `var`

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 3 – section – 11.13

ECMA 262 edition 3 – section – 12.2

Wrox *Instant JavaScript* ISBN 1861001-27-4 – page – 20

Associative array indexing (Advice)

Accessing array elements with strings as symbolic names.

Arrays and collections can be accessed using numeric indexing where each pocket is referred to by a number.

The array may be sparse and not all entries need to be assigned, but the length value will be set to one greater than the highest numbered entry. The first numbered entry is item 0.

You can also use strings instead of numbers. These string values can be specified literally or be passed with a variable.

These are all valid array element references:

```
myIndex = "three";
myArray[0] = "A";
myArray[100] = "B";
myArray["one"] = "C";
myArray['two'] = "D";
myArray[three] = "E";
```

See also:	Array index delimiter ([ ])

Associativity (Definition)

A direction of evaluation of an operator-driven expression.

The associativity of an operator indicates the order of evaluation of its operands. An operator with an associativity of left to right evaluates the expression in the operand to its left and then the one to the right. The alternative is right to left associativity.

atob() (Method)

Decode some base-64 encoded data.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N `atob(`*`aBase64String`*`)` N `myWindow.atob(`*`aBase64String`*`)`	
Argument list:	*`aBase64String`*	A string containing base 64 encoded data
See also:	`Window.btoa()`	

attachEvent() (Method)

A means of attaching events to windows and documents.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE `attachEvent(`*`anEventName`*`, `*`anEventHandler`*`)` IE *`myWindow`*`.attachEvent(`*`anEventName`*`, `*`anEventHandler`*`)`	
Argument list:	*`anEventHandler`*	A reference to an event handler function
	`anEventName`	The name of an event to be handled
See also:	.htc, `<STYLE>`, `Document.attachEvent()`, `Document.detachEvent()`, HTML Component, `onContentReady`, `onDocumentReady`, `Window.detachEvent()`, `Window.attachEvent()`	

Cross-references:

Wrox *Professional JavaScript* – page – 115

Attr object (Object/DOM)

This is implemented in MSIE as an `Attribute` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

Inherits from:	`Node` object	
JavaScript syntax:	-	*myAttr* = *myDocument*.createAttribute(*aName*)
Argument list:	*aName*	The name of the attribute to create

The DOM level 2 standard adds an `ownerElement` property to the `Attr` object specification. This is not yet supported in browsers.

See also:	`Attribute` object, `Document.createAttribute()`

Inheritance chain:

`Node` object

Attribute object (Object/DOM)

A DOM object that represents an HTML tag attribute.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Inherits from:	`Node` object	
JavaScript syntax:	-	*myAttribute* = *myAttributes*.*aPropertyName*
	-	*myAttribute* = *myAttributes*[*anIndex*]
	-	*myAttribute* = *myAttributes*[*aName*]
	-	*myAttribute* = *myDocument*.createAttribute(*aName*)
Argument list:	*aPropertyName*	The name of the tag attribute
	aName	An attribute name
	anIndex	A valid numeric reference to an element in the collection
Object properties:	`name`, `nodeName`, `nodeType`, `nodeValue`, `specified`, `value`	

This is used by the browser to maintain property values for HTML tag instantiated objects.

This object represents a single HTML tag attribute. The properties of this object indicate whether the tag attribute has been specified or not, and if it has, what the current value is.

The `Element` object should contain enough information for you to be able to determine the instantiating source tag name. The attributes can be inspected with a script and the complete source HTML reconstructed from a combination of the information supplied by the element and its associated `attributes` collection.

The `attributes` collection that belongs to an object also tells you what the expected complete set of attributes are for the tag, although this may not be completely reliable.

The example script demonstrates how you can make an `Attribute` object inspector with a fragment of JavaScript. These inspectors can be put into a library and called in for debugging when you are experiencing problems.

Note that the example overleaf does not work on Netscape 6.0 due to the use of the `all` property.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY alink=red>
<SCRIPT>
// An example attributes object inspector
myAttributeObject = document.all[3].attributes.aLink;
displayAttributes("BODY alink", myAttributeObject);
// Display attributes object
function displayAttributes(aTitle, anObject)
{
   document.write("<H3>");
   document.write(aTitle);
   document.write("</H3>");
   document.write("<TABLE BORDER=1 CELLPADDING=2><TR>");
   document.write("<TH>Description</TH>");
   document.write("<TH>Property</TH>");
   document.write("<TH>Value</TH></TR>");
   displayTableLine("Tag attribute name:", "name", anObject.name);
   displayTableLine("Tag attribute value:", "value", anObject.value);
   displayTableLine("DOM node name:", "nodeName", anObject.nodeName);
   displayTableLine("DOM node type:", "nodeType", anObject.nodeType);
   displayTableLine("DOM node value:", "nodeValue", anObject.nodeValue);
   displayTableLine("Specified flag:", "specified", anObject.specified);
   document.write("</TABLE>");
}
// Display a table line
function displayTableLine(aDescription, aProperty, aValue)
{
   document.write("<TR><TH ALIGN=LEFT>");
   document.write(aDescription);
   document.write("</TH><TD>");
   document.write(aProperty);
   document.write("</TD><TD>");
   document.write(aValue);
   document.write("</TD></TR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also: `Attr` object, `Attributes` object, `Document.createAttribute()`, `Element.getAttributeNode()`, `Element.removeAttribute()`, `Element.removeAttributeNode()`, `Element.setAttributeNode()`, `HasProperty()`, HTML tag attribute, `MutationEvent.attrChange`, `MutationEvent.attrName`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`name`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly
`nodeName`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`nodeType`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`nodeValue`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`specified`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly
`value`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Inheritance chain:

`Node` object

Attribute.name (Property)

The name of the HTML tag attribute this object represents.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myAttribute`*`.name`

The name of the HTML tag attribute that this object represents is reflected here. This is the same value as the property name within the `Attributes` array that refers to this object. You can use that value associatively or as if it were a property name.

This value is also the same as a property name belonging to the `Element` object that represents the HTML tag that this is an attribute of.

Property attributes:

`ReadOnly`.

Attribute.nodeName (Property)

Another alias for the `name` property of an `Attribute` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myAttribute*.nodeName

This is provided to support some previous usage that accessed the tag name under the `nodeName` property. The same value is available in the `name` property of the `Attribute` object. It may contain some values that the `name` property does not support.

The following values may be seen in this property:

- The tag name, also visible via the `tagName` property of the owning object
- The attribute name for those nodes that are `Attribute` objects
- The value `#text` for nodes that encapsulate a block of raw text and are `textNode` objects

See also:	`Element.tagName`, `textNode` object

Attribute.nodeType (Property)

Part of the internal document hierarchy management within MSIE.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myAttribute*.nodeType

The node hierarchy is built from objects that represent a variety of different kinds of Document Object Model content.

The two principle node types are HTML Element nodes (value 1) and Text content nodes (value 3).

Here is a list of the available node types:

Constant:	Type:	Description:
undefined	null	A member of the attributes collection
ELEMENT_NODE	1	HTML element object node
ATTRIBUTE_NODE	2	HTML tag attribute object
TEXT_NODE	3	Text object node
CDATA_SECTION_NODE	4	CDATA section
ENTITY_REFERENCE_NODE	5	Entity reference

Table continued on following page

Constant:	Type:	Description:
ENTITY_NODE	6	Entity node
PROCESSING_INSTRUCTION_NODE	7	Processing instruction node
COMMENT_NODE	8	Comment node
DOCUMENT_NODE	9	Document object
DOCUMENT_TYPE_NODE	10	Doctype object
DOCUMENT_FRAGMENT_NODE	11	Document fragment node
NOTATION_NODE	12	Notation node

See also:	Node object

Attribute.nodeValue (Property)

Another name for the value property of this object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myAttribute*.nodeValue

The value portion of the ATTRIBUTE="aValue" construct. Knowing the name and value of an attribute provides sufficient information to reconstruct the source HTML code from scratch. Some nodeValues may not be defined by ATTRIBUTE HTML tag attributes, but may be the content of the tag itself.

If the object that the attributes are associated with is a TxtNode object, then the nodeValue should return the textual content encapsulated by that object. In that case, the nodeValue cannot be modified although it may be writable for other object types.

If the nodeType is an attribute then the nodeValue reflects its HTML tag attribute value or null if it has not been defined.

If the nodeType is an HTML Element object, then the nodeName should be used to determine which tag it encapsulates. In that case the nodeValue should yield a null.

Attribute.specified (Property)

Whether the value has been specified or not.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`myAttribute`*`.specified`

This flag indicates whether the attribute is defined or not. This property is updated automatically when you change the value of a property belonging to an `Element` object.

The attribute contains a meaningful value only if this property is set to `true`. If it is set to `false`, the other properties belonging to the `Attributes` object don't hold any useful information.

The result is `true` if the value is specified by an HTML tag attribute and `false` if it is not currently specified.

Property attributes:

`ReadOnly`.

Attribute.value (Property)

The value of the HTML tag attribute if it has been specified.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myAttribute`*`.value`

This property reflects the value defined by the parent HTML tag attribute. It is part of the DOM support for HTML tag attributes.

If an HTML tag attribute is specified, the attribute flag for that property of the HTML tag's object will be set `true`. The actual value of the HTML tag attribute will be stored in this property. This `Attribute.value` property gets updated automatically if the owner `Element` object's property value is changed. Internally the two property accessors probably refer to the same storage location.

Attributes object (Object/DOM)

A sub-class of the `Array` object that contains a set of `Element` object attributes. This is a collection of all attribute objects that apply to an element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myAttributes* = *myElement*.attributes
Object properties:	`length`	

The `Attributes` array object is associated with an `Element` object as a container for a set of `Attribute` objects each of which relates to a property of the `Element` object. This is the correct implementation of the DOM specified `Attr` object class.

Not all `Element` object properties have an `Attribute` object, but those that do have related to HTML tag attributes. Thus the `Attributes` array corresponds to the HTML tag attributes for a tag.

The Attributes array has a length property so that you can enumerate all of the attributes of its instantiating HTML tag. It also has as many additional properties as are required to reflect each HTML tag attribute for that HTML tag. The properties reflected from the HTML tag attributes are named consistently with the HTML source text.

Properties are reserved to support event handlers and other tag attributes and so from the `Attributes` array for a particular `Element` object, you can establish what the supported features are for the HTML tag it represents. This means that the `length` property will vary from object to object.

The `Attributes` array seems to contain some properties that correspond to the imaginary HTML generic `Element` class. Although this is not really a genuine object class, it is a convenient way of documenting HTML object behaviors where they are common across a range of objects. The `Attributes` array does not support a complete set of properties that correspond to the `Element` class and therefore it is not true to say it inherits from that class.

The example script shows how you can inspect the attributes of an object. In this example, the attributes of a `<BODY>` tag are exposed. Because they are enumerable, you can determine what properties and what events the object instantiated by the `<BODY>` tag can respond to. Note that the example does not work on Netscape 6.0 due to the use of the `all` property.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY alink=red vlink="blue" leftmargin="100">
<TABLE BORDER=1 CELLPADDING=2>
<SCRIPT>
myAttributesObject = document.all[3].attributes;
displayTableLine("Object class:", myAttributesObject, "");
```

```
displayTableLine("Number of attributes:", myAttributesObject.length, "");
for(myEnumerator=0; myEnumerator<myAttributesObject.length; myEnumerator++)
{
   myAttrib = myAttributesObject[myEnumerator];
   displayTableLine("Attribute ("+ myAttrib.nodeName +"):", myAttrib.specified,
myAttrib.nodeValue);
}

// Output one line of a table
function displayTableLine(aHeading, aFlag, aValue)
{
   document.write("<TR>");
   document.write("<TH ALIGN=LEFT>");
   document.write(aHeading);
   document.write("</TH>");
   document.write("<TD>");
   document.write(aFlag);
   document.write("</TD>");
   document.write("<TD>");
   document.write(aValue);
   document.write("</TD>");
   document.write("</TR>");
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	`Attribute` object, `Attributes.length`, `Collection` object, `Element` object, `Element.attributes[]`, `Element.removeAttribute()`, `HasProperty()`, `HTML` object, HTML tag attribute

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`length`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly

Attributes.length (Property)

The number of tag attributes supported in this `Attributes` array.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myAttributes*`.length`

The number of attributes supported by the HTML tag that owns this `Attributes` object.

This is the principle property of an `Attributes` object. Most others depend on the object that is represented.

See also:	`Attributes` object, `Collection.length`

Property attributes:

`ReadOnly`.

ATVEF (Standard)

Advanced Television Enhancement Forum.

This extract from the ATVEF standard describes in outline the aims and scope of this web and TV convergence project. You should consult the specification for a complete description of how this is to be accomplished. There are several manufacturers already building and deploying these systems on a variety of broadcast mediums.

The Advanced Television Enhancement Forum (ATVEF) is a group of people from the broadcast TV and Internet industries who are working to specify a single public standard for delivering interactive television experiences. The intention is that these should be authored once using a variety of tools and deployed to a range of television, set-top, and PC-based receivers.

The Enhanced Content Specification defines the fundamental requirements that are necessary to enable creation of HTML-enhanced television content. This goes beyond normal Internet-based delivery to describe how it can be reliably broadcast across any network to any compliant receiver. Because the broadcast requires that there is no bidirectional link, some changes to the delivery protocols are outlined.

The ATVEF specification for enhanced television programming uses existing Internet technologies. It describes how to deliver enhanced TV programming over both analog and digital video systems using terrestrial, cable, satellite, and Internet networks. The specification can be used in both one-way broadcast and two-way video systems, and is designed to be compatible with all international standards for both analog and digital video systems.

See also:	Interpret, Liberate TV Navigator, Microsoft TV, URL, WebTV

Web-references:

http://atvef.com/library/spec1_1a.html

Aural style sheets (Definition)

The CSS standard describes style properties for spoken text.

The aural style properties allow the control of spoken voice and other sound effects to be assigned to element objects so that as they are displayed, their content may be spoken or read out to the user. This then makes the World Wide Web more accessible to sight-impaired users.

So far, not much of this capability has found its way into the currently available web browsers.

There are many issues that have not yet been addressed with this aspect of style sheets. For example, controlling multi-lingual spoken text and dates may be somewhat problematic. Certainly the locale that the browser is operating in may be used to select a national language variant for the spoken word.

Warnings:

- This facility is not yet supported by any of the browsers.

AuthentiCode (Security related)

This is a security model that applies digital signatures to ActiveX objects in MSIE.

Warnings:

- This technique does not currently support signed scripts in MSIE and only applies to `ActiveX` objects.

See also:	Security policy, Signed scripts

Automatic semi-colon insertion (Definition)

The action of adding semi-colons where they have been omitted.

Availability:	ECMAScript edition – 2

A semi-colon explicitly placed in the source text must terminate certain statements. Your JavaScript interpreter may help by adding some automatically, but this may not work as you expect. As they say, "Your mileage may vary".

Semi-colons are used to explicitly terminate certain keywords so that the parser can determine exactly where the fragment of code begins and ends. The semi-colon removes the ambiguity about how a piece of code is intended to execute.

Line terminators greatly affect the automatic semi-colon insertion process.

The following statements must have trailing semi-colons:

- `empty` statement
- `variable` statement
- `expression` statement
- `continue` statement
- `break` statement
- `return` statement
- `throw` statement

There are cases where the the interpreter will automatically insert semi-colons as needed. You won't see them in the script source, but the interpreter knows they should be there. You should not rely on the interpreter doing your work for you. For example, semi-colons are never added inside `for` statement headers. Here are some instances of how the browser deals with automatic semi-colon insertion:

- Semi-colons are automatically placed before curly braces (}) that close code blocks if necessary.
- A semi-colon is added at the end of a script source text if necessary to parse the source as a complete program.
- Semi-colons are added to prevent accidental postfix increment or decrement operations. Postfix ++ or -- operators should be on the same line as the operand to which they apply. Actually it is good practice for there to be no whitespace between them.
- Semi-colons are added after the `return` statement when it is the last statement on a line. An expression to be evaluated as part of a `return` statement should be placed adjacent to it. It is good practice to form the return as if it were a function, enclosing the expression in parentheses:

```
return(expression);
```

- This is unaffected by automatic semi-colon insertion even though it is syntactically incorrect:

```
for (a; b)
```

- This is transformed:

```
returna + b
```

And becomes:

```
return;a + b;
```

However, `a + b` is not returned as a result because the line terminator separates them from the `return` statement.

- People take a great many liberties with the formatting of `if…else` constructions. This won't get fixed:

```
if(a > b)else c = d
```

- This won't get fixed either:

```
a = b + c(d + e).print()
```

It doesn't get fixed because the parentheses look like a function call.

Warnings:

- Careful programmers always put semi-colons in. If you come from a C or Java background, this may be instinctive, but otherwise you should develop the habit so that it becomes instinctive.

See also:	Free-format language, Lexical convention, Line terminator, Semi-colon (;)

Cross-references:

ECMA 262 edition 2 – section – 7.8

ECMA 262 edition 3 – section – 7.9

O'Reilly *JavaScript Definitive Guide* – page – 28

Wrox *Instant JavaScript* – page – 17

Automation object (Object/JScript)

An object created in the JScript environment for connecting to other applications within the host environment.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`myAutomation = GetObject(aLocation)`
	IE	`myAutomation = GetObject(aLocation, anObjectType)`
	IE	`myAutomation = GetObject(aLocation!aSubObject)`
	IE	`myAutomation = GetObject (aLocation!aSubObject, anObjectType)`
Argument list:	`anObjectType`	What sort of application and object class type to be created
	`aLocation`	A path to the file for the object to be instantiated
	`aSubObject`	A fragment identifier for a sub-object within the file

See also:	`ActiveXObject` object, `GetObject()`

B object (Object/HTML)

An object that represents the font style controlled by the <B> HTML tag.

Availability:	JScript –3.0 Internet Explorer –4.0		
Deprecated Usage:	Yes		
Inherits from:	`Element` object		
JavaScript syntax:	IE	*myB* = *myDocument*.all.*anElementID*	
	IE	*myB* = *myDocument*.all.tags("B")[*anIndex*]	
	IE	*myB* = *myDocument*.all[*aName*]	
	-	*myB* = *myDocument*.getElementById(*anElementID*)	
	-	*myB* = *myDocument*.getElementsByName(*aName*)[*anIndex*]	
	-	myB = *myDocument*.getElementsByTagName("B")[*anIndex*]	
HTML syntax:	<B> ... </B>		
Argument list:	*anIndex*		A valid reference to an item in the collection
	aName		The `name` attribute of an element
	anElementID		The `ID` attribute of an element
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart		

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

See also:	`Element` object

back() (Method)

Perform the same action as pressing the `[BACK]` button in the toolbar.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`back()`
	-	*myWindow*`.back()`

See also:	`History.back()`, `Window.forward()`, Window.back()

Background object (Object/browser)

A background image object associated with a Netscape Navigator layer.

Availability:	JavaScript – 1.2 Netscape Navigator version – 4.0	
JavaScript syntax:	N	*myBackground* = *myLayer*.background
Object properties:	src	

This object is used with a layer in Netscape Navigator and its properties correspond with properties of the `Image` object in Netscape Navigator.

See also:	`Background.src`,`BODY` object,`Image` object,`Layer.background`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
src	1.2 +	-	4.0 +	-	-	-	-

Background.src (Property)

The source location of an image to be associated with a layer and used as its background image.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myBackground*.src

This corresponds to the src property of a Netscape Navigator Image object. It defines the URL of an image to load into the background of a layer, as these are scriptable in Netscape Navigator whereas the background image of a document object is not.

See also:	`Background` object, `BODY.background`, `BODY.bgProperties`, `JSSTag.backgroundImage`,`style.background`

Back-quote (`) (External code call)

Call some external code during server-side execution.

The back-quote substitutions operate much like you may have seen them work in command-line shells and Perl interpreters. The text enclosed inside the back-quotes is parsed out from the HTML and is then executed as JavaScript.

ASP provides a means of substituting the output of JavaScript code into a block enclosed in `<% ... %>` markers which does a similar thing.

This allows us to include fragments of JavaScript into an HTML page and expect them to be parsed server-side.

This is somewhat analogous to JavaScript entities but they operate at the client-side.

The server-side example wraps its result inside quote symbols so that the HTML tag attribute syntax is preserved intact.

Example code:

```
<HTML>
<BODY>
<FORM>
<INPUT TYPE="text" VALUE=`server.hostname;`>
</FORM>
</BODY>
</HTML>
```

See also:	JavaScript entity, Netscape Enterprise Server

Bar object (Object/Navigator)

An object used to hold properties for toolbars, location bars etc.

Availability:	JavaScript – 1.2 Netscape Version – 4.0	
JavaScript syntax:	N	*myBar* = locationbar
	N	*myBar* = menubar
	N	*myBar* = *myWindow*.locationbar
	N	*myBar* = *myWindow*.menubar
	N	*myBar* = *myWindow*.personalbar
	N	*myBar* = *myWindow*.scrollbars
	N	*myBar* = *myWindow*.statusbar
	N	*myBar* = *myWindow*.toolbar
	N	*myBar* = personalbar
	N	*myBar* = scrollbars
	N	*myBar* = statusbar
	N	*myBar* = toolbar
Object properties:	visible	

This object is used to represent various items of window furniture (otherwise called chrome or adornments) in Netscape Navigator. It isn't supported by MSIE although the control facilities it offers are available when a new window is created with the `window.open()` method.

It only has one usable property. That is the `visible` property, which can be set to a Boolean value. Some early documentation referred to this as the `visibility` property but that is the wrong property name.

See also:	`Bar.visible`, `Window.locationbar`, `Window.menubar`, `Window.personalbar`, `Window.scrollbars`, `Window.statusbar`, `Window.toolbar`

Property	JavaScript	JScript	N	IE	Opera	Notes
`visible`	1.2 +	-	4.0 +	-	-	-

Bar.visibility (Pitfall)

An erroneous name for the `visible` property of a `Bar` object.

Warnings:

- Some reference works refer to the `visibility` property of the `Bar` object, possibly due to early prototype versions of the Netscape browser or in an attempt to document forthcoming features of the browser. In between publishing and release of the browser, the property changed its name to the `visible` property.
- You may even then have some difficulty in getting it to work on some platforms but you do need to make sure you are trying to set the correct property value when changing the visibility of `Bar` objects.

See also:	`Bar.visible`

Bar.visible (Property)

A flag indicating whether the bar that this object represents is visible.

Availability:	JavaScript – 1.2 Netscape Navigator version – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myBar*`.visible`
	N	*myBar*`.visible = `*aBoolean*
Argument list:	*aBoolean*	A switch value to control the visibility of a window control item

Setting this property to `true` makes the bar visible. Setting `false` hides the bar.

You must have been granted the `UniversalBrowserWrite` privilege to be able to set this property value.

See also:	`Bar` object, `Bar.visibility`

Barn() (Filter/transition)

A transition effect with the appearance of barn doors opening or closing.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

Filter – `Barn()`

BASE object (Object/HTML)

Represents the `<BASE>` HTML tag that describes a base URL for the document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myBASE* = *myDocument*.all.*anElementID*
	IE	*myBASE* = *myDocument*.all.tags("BASE")[*anIndex*]
	IE	*myBASE* = *myDocument*.all[*aName*]
	-	*myBASE* = *myDocument*.getElementById(*anElementID*)
	-	*myBASE* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myBASE* = *myDocument*.getElementsByTagName ("BASE")[*anIndex*]
HTML syntax:	`<BASE>`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The name attribute of an element
	anElementID	The `ID` attribute of an element
Object properties:	`href,target`	
Event handlers:	`onClick,onDblClick,onHelp,onKeyDown,onKeyPress, onKeyUp,onMouseDown,onMouseMove,onMouseOut, onMouseOver,onMouseUp`	

The `<<BASE>>` tag must appear inside the `<<HEAD>>` block of a document and is used to define a base URL for the document, this can be useful if the document is not served from the same server that subsequent pages need to be served from.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`href`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`target`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

BASE.href (Property)

The URL defined by the `<<BASE>>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myBASE`*`.href`

The URL to be used as a base for any relative URLs in the remainder of the document.

BASE.target (Property)

The target window or frame defined by the `<<BASE>>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape Navigator version – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myBASE*.target

The target window or frame to be added to any relative (or non targeted) URL values in the remainder of the document.

You can assign a new value to this property so that any URLs that are built by the browser with a relative location will be directed to a different window or frame.

Here are some example target values:

- `_parent`
- `_self`
- `_top`
- `_blank`
- Window name
- Frame name

See also:	`Anchor.target`, `Form.target`, `Location.target`, `Map.target`, `Url.target`

BASEFONT object (Object/HTML)

A `<<BASEFONT>>` HTML tag is represented by this object and defines some generic font information to be used as a default in this page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape Navigator version – 6.0
Inherits from:	`Element` object

JavaScript syntax:	IE	*myBASEFONT* = *myDocument*.all.*anElementID*
	IE	*myBASEFONT* = *myDocument*.all.tags("BASEFONT")[*anIndex*]
	IE	*myBASEFONT* = *myDocument*.all[*aName*]
	-	*myBASEFONT* = *myDocument*.getElementById(*anElementID*)
	-	*myBASEFONT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myBASEFONT* = *myDocument*.getElementsByTagName("BASEFONT")[*anIndex*]
HTML syntax:	<BASEFONT>	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The name attribute of an element
	anElementID	The ID attribute of an element
Object properties:	color, face, size	
Object methods:	getAttribute()	
Event handlers:	onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp	

Historically web developers will have used the <FONT> tag to set the attributes of blocks of text. Latterly, they will be using style sheets to control this.

The <BASEFONT> tag provides a way to set the font presentation style from the position of this tag to the end of the document unless overridden by further <BASEFONT> tags or <FONT> settings.

See also:	Element object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
color	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
face	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
size	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
getAttribute()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

BASEFONT.color (Property)

The default color of text affected by the `<BASEFONT>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape Navigator version – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBASEFONT.color`

The color of text affected by this `BASEFONT` object will be defined in this property.

The color can be specified in the normal way according to the HTML color specifiers.

See also:	Color value, `FONT. color`

BASEFONT.face (Property)

The default font face for text affected by the <BASEFONT> HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape Navigator version – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	myBASEFONT.face

The font face to be used for subsequent text is defined by this property. It is appropriate to define a list of font faces in priority order in the normal way. The browser will use the first one it encounters that it has available.

BASEFONT.size (Property)

The default size of text affected by the <<BASEFONT>> HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape Navigator version – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	myBASEFONT.size

The size of text rendered by the browser under control of the BASEFONT object is controlled by this property. Absolute and relative sizes are supported in the normal way.

Basic type (Definition)

Another name for the native types supported by the interpreter.

See also:	Native object, Primitive value

BasicImage() (Filter/visual)

Controls over the basic image display attributes of the containing HTML `Element` object.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

Filter – `BasicImage()`

BDO object (Object/HTML)

An object representing the `<BDO>` HTML tag for supporting bidirectional text algorithms.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myBDO* = *myDocument*.all.*anElementID*
	IE	*myBDO* = *myDocument*.all.tags("BDO") [*anIndex*]
	IE	*myBDO* = *myDocument*.all[*aName*]
	-	*myBDO* = *myDocument*.getElementById (*anElementID*)
	-	*myBDO* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myBDO* = *myDocument*.getElementsByTagName("BDO")[*anIndex*]
HTML syntax:	`<BDO> ... </BDO>`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The `name` attribute of an element
	anElementID	The `ID` attribute of an element
Object properties:	`dir`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

This is the `Bi-Directional Override` object. The `LANG` and `DIR` attributes of HTML tags in the document will cover most eventualities but there may be times when you need to explicitly override the direction of text flow.

Usage of this is likely to be confined to scripts that operate in multiple-language environments and on pages containing text in more than one language.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`dir`	-	5.0 +	-	5.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onDblClick`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onHelp`	-	5.0 +	-	5.0 +	-	-	-	Warning
`onKeyDown`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onKeyPress`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onKeyUp`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onMouseDown`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onMouseMove`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onMouseOut`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onMouseOver`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
`onMouseUp`	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

BDO.dir (Property)

The direction attribute of the `<BDO>` HTML tag.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myBDO.dir`

The `dir` property may be set to indicate a left to right or right to left parsing direction.

This is part of the localization support and represents the contents of the `DIR=". . ."` tag attribute.

If you assign a value to this property it is case-sensitive and must be either "`ltr`" or "`rtl`".

This property works in conjunction with the `lang` property to control the direction of text flow.

See also:	`Element.dir`,`NOFRAMES.dir`,`NOSCRIPT.dir`

BeanConnect (Definition)

A Netscape Communications technology for interconnecting Java applets (Beans).

See also:	Java,Java exception events

Refer to:

LiveConnect

Behavior (Definition)

Implementations respond to different constructs according to their behavior.

The ECMAScript standard defines how an implementation should react to a language construct. Other non-ECMA-compliant implementations may behave in the same way most of the time and may deviate from the standard at others.

When the implementation conforms to the standard, its behavior is predictable according to the definitions of the standard. When an implementation is not conformant, it may behave according to one of the following abnormal behavior models:

- Unspecified behavior
- Undefined behavior
- Implementation-defined behavior
- Locale-specific behavior

Another meaning for the word behavior in the context of JavaScript is the way that MSIE supports the addition of JavaScript functionality to style definitions. This is covered under the descriptions of the `addBehavior`() and `removeBehavior`() methods that belong to the `Element` object.

See also:	Compliance, `Element.addBehavior()`,`Element.filters[]`, `Element.removeBehavior()`, Implementation-defined behavior,Locale-specific behavior, Undefined behavior, Unspecified behavior

BGSOUND object (Object/HTML)

An object representing a `<BGSOUND>` HTML tag that defines an audio track to play while the page is displayed.

Availability:	JScript – 3.0 Internet Explorer – 4.0 (as HTML in IE 3.0)
Inherits from:	`Element` object

JavaScript syntax:	IE	*myBGSOUND* = *myDocument*.all.*anElementID*
	IE	*myBGSOUND* = *myDocument*.all.tags("BGSOUND")[*anIndex*]
	IE	*myBGSOUND* = *myDocument*.all[*aName*]
	-	*myBGSOUND* = *myDocument*.getElementById (*anElementID*)
	-	*myBGSOUND* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myBGSOUND* = *myDocument*.getElementsByTagName ("BGSOUND")[*anIndex*]
HTML syntax:	<BGSOUND>	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The name attribute of an element
	anElementID	The ID attribute of an element
Object properties:	balance,loop,src,volume	
Event handlers:	onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp	

This object is instantiated by the <BGSOUND> HTML tag and represents a sound effect that is to be played in the background. As the BGSOUND object is created during document loading and requires that a sound file be downloaded, there may some noticeable delay before the sound starts to play.

See also:	Element object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
balance	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
loop	-	3.0 +	-	4.0 +	-	-	-	-
src	-	3.0 +	-	4.0 +	-	-	-	-
volume	-	3.0 +	-	4.0 +	-	-	-	ReadOnly

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

BGSOUND.balance (Property)

The stereo balance of the background sound.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`myBGSOUND.balance`

The relative volume of the left and right channels will be adjusted according to the value of this property. This provides a limited amount of control over the apparent direction of the sound source.

Creative use of the balance property may be tied in to the horizontal scrolling of a page for creating virtual reality effects.

Much more sophisticated control is available through the aural style sheet properties, although these are not yet properly supported by browsers.

See also:	`style.azimuth`

Property attributes:

`ReadOnly`.

BGSOUND.loop (Property)

Whether the background sound should loop when it gets to the end.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`myBGSOUND.loop`

This indicates the number of times that the sound should play before stopping.

BGSOUND.src (Property)

The URL that the background sound file can be fetched from.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myBGSOUND.src`

The sound will be loaded from this location while the page is being constructed. There may be some delay between requesting the sound and being able to play it.

You can define a new value here to load a different sound and play it in the background.

BGSOUND.volume (Property)

The volume setting at which the background sound should play.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myBGSOUND`*`.volume`

The volume setting of the background can be modified by this property.

The actual perceived volume may depend on other factors. If the sound has been digitized at an unusually low volume, you may need to raise the volume setting quite high. This may yield a very noisy sound as you will also be increasing the ambient noise in the sampled sound. Digitizing is a complex activity but you should always strive for the highest possible signal to noise ratio.

Other factors that may affect the apparent volume would be the user preference settings in the computer. There may also be system controls for blending and mixing sound sources and these may be set to unhelpful values.

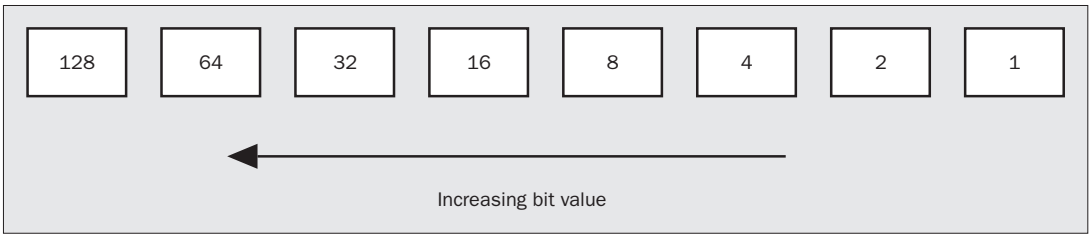
This property can obviously only control the source volume of the sound generated by the browser.

Property attributes:

`ReadOnly`.

Big endian (Definition)

A bit ordering standard for some CPU models.



Refer to:

byte

BIG object (Object/HTML)

An object that represents the font style controlled by the <<BIG>> HTML tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Deprecated Usage:	Yes	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myBIG* = *myDocument*.all.*anElementID*
	IE	*myBIG* = *myDocument*.all.tags("BIG") [*anIndex*]
	IE	*myBIG* = *myDocument*.all[*aName*]
	-	*myBIG* = myDocument.getElementById (anElementID)
	-	*myBIG* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myBIG* = *myDocument*.getElementsByTagName ("BIG")[*anIndex*]
HTML syntax:	`<BIG> ... </BIG>`	
Argument list:	`anIndex`	A valid reference to an item in the collection
	`aName`	The `name` attribute of an element
	`anElementID`	The `ID` attribute of an element
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onDblClick`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, `Node` object

Refer to:

`Element` object

Binary bitwise operator (Definition)

An operator that applies in a bitwise fashion.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

A binary bitwise operator converts its operands to 32 bit values and performs the operation on each corresponding bit in the two values.

Warnings:

- The result of a bitwise expression is a 32 bit binary value and should not be confused with the Boolean value returned by a logical operator.

See also:	Bitwise AND (&), Bitwise operator, Bitwise OR (|), Bitwise XOR (^)

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.10

Binary logical operator (Definition)

An operator that works with Boolean `true` or `false` values.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

Binary logical operators test a pair of Boolean values according to logical rules. If necessary, JavaScript will convert the operands that are passed to the expression into Boolean values before testing them. You should consult the `toBoolean` rules for each type of object being passed to ensure that values are cast in a way that you expect.

The resulting value of a binary logical expression may be coerced to another data type on return.

There is no logical XOR operator. It can be simulated though by testing two Boolean values for inequality, since that is going to occur when either is one and the other is zero; the inequality will not test `true` if both are one or both are zero.

Warnings:

- This is not to be confused with the bitwise operators, which yield a 32-bit integer value instead of the Boolean value yielded by a logical expression.

See also:	Logical AND (&&), Logical operator, Logical OR (||)

Cross-references:

ECMA 262 edition 2 – section – 11.11

ECMA 262 edition 3 – section – 11.11

Binary operator (Definition)

An operator that works with two operands.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

Binary operators require two operands and with them form an expression. The operator determines the kind of expression.

Here is a list of the binary operators supported by JavaScript:

Operator	Description
`!=`	NOT equal to
`%`	Remainder
`%=`	Remainder and assign to an LValue
`&`	Bitwise AND
`&&`	Logical AND
`&=`	Bitwise AND and assign to an LValue
`*`	Multiply
`*=`	Multiply and assign to an LValue
`+`	Add
`+`	Concatenate string
`+=`	Add and assign to an LValue
`-`	Subtract
`-=`	Subtract and assign to an LValue
`/`	Divide
`/=`	Divide and assign to an LValue
`<`	Less than
`<<`	Bitwise left shift
`<<=`	Bitwise shift left and assign to an LValue
`<=`	Less than or equal to
`=`	Simple assignment to an LValue
`==`	Equal to
`>`	Greater than
`>=`	Greater than or equal to
`>>`	Bitwise shift right
`>>=`	Bitwise shift right and assign to an LValue
`>>>`	Bitwise shift right (unsigned)

Table continued on following page

Operator	Description
>>>=	Bitwise shift right (unsigned) and assign to an LValue
^	Bitwise XOR (exclusive OR)
^=	Bitwise exclusive XOR and assign to an LValue
|	Bitwise inclusive OR
|=	Bitwise inclusive OR and assign to an LValue
||	Logical OR

See also:	Multiplicative operator, Operator, Ternary operator

Cross-references:

ECMA 262 edition 2 – section – 11.5

ECMA 262 edition 3 – section – 11.6

ECMA 262 edition 3 – section – 11.7

ECMA 262 edition 3 – section – 11.8

ECMA 262 edition 3 – section – 11.9

ECMA 262 edition 3 – section – 11.10

ECMA 262 edition 3 – section – 11.11

ECMA 262 edition 3 – section – 11.13

Binding (Definition)

Binding is used to resolve identifiers via the scope chain.

Availability:	ECMAScript edition – 2

Binding is the process of locating the appropriate object or property where a value is stored for a particular identifier.

The binding process uses the scope chain belonging to the current execution context to locate the earliest matching item according to the inheritance rules.

See also:	Identifier resolution

Cross-references:

ECMA 262 edition 2 – section – 10.1.4

ECMA 262 edition 3 – section – 10.1.4

Bit (Definition)

A binary digit.

A Boolean value can be represented as a bit. Since a bit can maintain exactly two states (true or false), the two map very well to one another. Strictly speaking a Boolean value may yield an undefined state as well.

A continuous series of 8 bits forms a byte and 16 form a word. In JavaScript, 16 bit values tend to be the smallest that you operate with and correspond to a single character in a Unicode string. However, you can probably represent most characters that you want to use in the English language with only 8 bits. In fact only 7 bits are sufficient to describe your script source text in an ASCII representation.

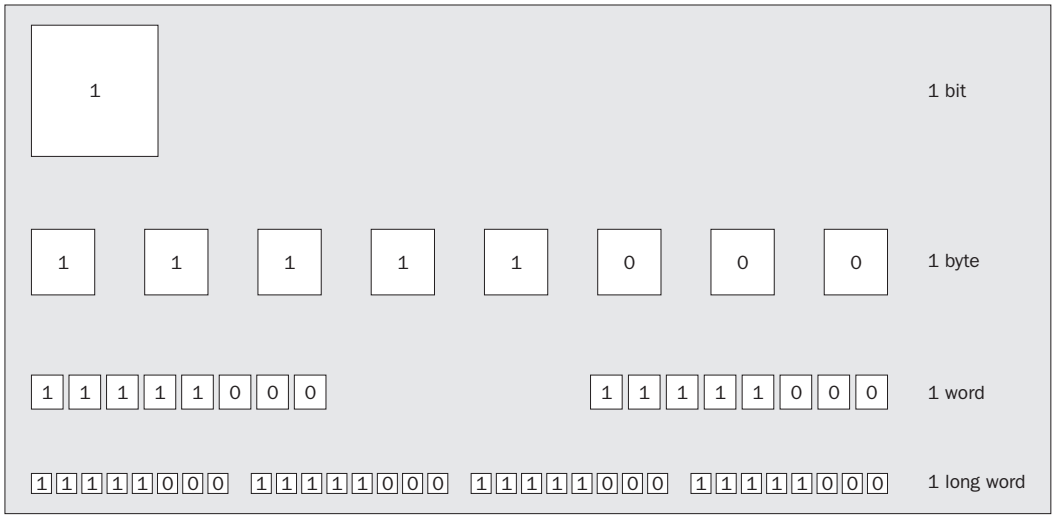
Bit manipulation of character values allows you to convert between upper and lower case. The `String.toUpperCase()` and `String.toLowerCase()` methods allow you to convert specifically to the case you want, but if the current case is unknown and you simply want to toggle the case of a character, the difference between 'A' and 'a' is a single bit.

In most cases, you won't be operating with binary digits in JavaScript-based projects. However, the language is quite capable of working with bit patterns provided you understand how they work.

Although you cannot store an individual bit on its own, you can keep collections of 32 of them in a Number value.

In C language you operate on these using Bit-Fields. JavaScript does not support bit-fields but the sort of things you do with them can be simulated.

This is likely to be of most use to people developing scripts for use in embedded interpreters and of less use to browser script developers.



Example code:

```
// Demonstrate bit inversion to change character case
myString = "AbCdEfGh";
myLength = myString.length;
document.write("Original source string : ");
document.write(myString);
document.write("<BR>");
document.write("<BR>");
document.write("<TABLE BORDER=1><TR><TH>");
document.write("Orig char</TH><TH>");
document.write("Char code</TH><TH>");
document.write("Bit inverted<BR>char code</TH><TH>");
document.write("New char</TH></TR>");
for(myEnum = 0; myEnum < myLength; myEnum++)
{
   myChar     = myString.charAt(myEnum);
   myCharCode = myString.charCodeAt(myEnum);
   myNewCharCode = myCharCode ^ 32;

   document.write("<TR><TD>");
   document.write(myChar);
   document.write("</TD><TD>");
   document.write(myCharCode);
   document.write("</TD><TD>");
   document.write(myNewCharCode);
   document.write("</TD><TD>");
   document.write(String.fromCharCode(myNewCharCode));
   document.write("</TD></TR>");
}
document.write("</TABLE>");
```

See also:	Bit-field, `String.toLocaleLowerCase()`, `String.toLocaleUpperCase()`,`String.toLowerCase()`, `String.toUpperCase()`

Bit-field (Definition)

A collection of binary digits.

Although JavaScript does not support bit-fields, you can perform many binary operations on patterns of bits by using the bitwise operators and various simple mathematical expressions to simulate other bit manipulation operators that are not provided as part of the standard.

Op	Description
~	Bitwise complement (NOT)
&	Bitwise AND
<<	Bitwise left shift
>>	Bitwise right shift
>>>	Bitwise right shift (unsigned)

Op	Description
|	Bitwise inclusive OR
^	Bitwise XOR (exclusive OR)
&=	Bitwise AND and assign to an LValue
|=	Bitwise inclusive OR and assign to an LValue
^=	Bitwise exclusive XOR and assign to an LValue
<<=	Bitwise shift left and assign to an LValue
>>=	Bitwise shift right and assign to an LValue
>>>=	Bitwise shift right (unsigned) and assign to an LValue

The bits are individually weighted according to their position relative to the least significant digit.

The single bit at the extreme right-hand end is defined by the integer value 1. The next significant bit is derived by using a zero-based indexing scheme to raise 2 to the power of its index position. Thus 2 raised to the power 0 is 1. The value 2 raised to the power 1 is 2 and thus the values proceed like this, moving from right to left:

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 etc.

To build bit masks containing a set bit for several positions, simply add the component bit values together. Thus a mask that includes all the four least significant bits in a value is equal to:

1 + 2 + 4 + 8

Here are some other useful mask values (note that we only show 8 bit values here to demonstrate the concept):

Mask	Value	Description
0000 0001	1	Least significant bit
0000 1111	15	Least significant nibble
0010 0000	32	ASCII upper/lowercase character bit
0101 0101	85	Simple encryption pattern for XOR
0111 1111	127	Valid ASCII character mask
1111 0000	240	Most significant nibble
1111 1111	255	Low 255 UNICODE character set mask

See also:	Bit,Bitwise AND (&), Bitwise AND then assign (&=), Bitwise expression,Bitwise NOT – complement (~), Bitwise operator,Bitwise OR (|), Bitwise OR then assign (|=), Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator,Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), Bitwise XOR (^), Bitwise XOR and assign (^=),Expression

Bitwise AND (&) (Operator/bitwise)

Bitwise AND of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 & anOperand2`
Argument list:	*`anOperand1`* *`anOperand2`*	A binary bit pattern Another binary bit pattern

The result is the bitwise AND of both binary bit pattern values.

This operator performs a bit by bit AND of the 32-bit value derived from both operands. Effectively, each corresponding bit pair has a logical AND applied to it.

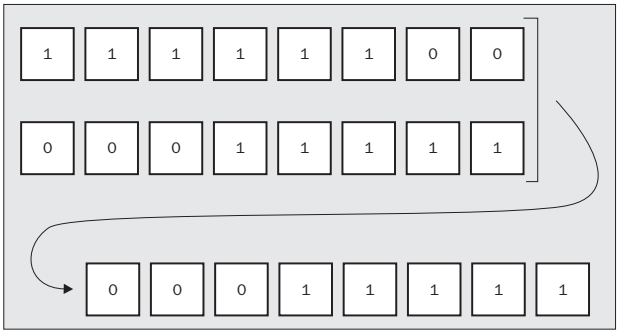
The truth table shows the result of this operator for two Boolean primitive values:

A	B	AND
false	false	false
false	true	false
true	false	false
true	true	true

Where a corresponding bit is 1 in both values, a 1 bit is inserted into the result otherwise the value is zero.

The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.



Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myValue1 = 0xFFFF;
myValue2 = 0xFF00;
myValue3 = myValue1 & myValue2;

document.write("Val 1 : " + binary32(myValue1) + "<BR>");
document.write("Val 2 : " + binary32(myValue2) + "<BR>");
document.write("AND : " + binary32(myValue3) + "<BR>");

// Binary convertor (ignore sign bit on MSIE)
function binary32(aValue)
{
      myArray = new Array(32);

      for(myEnum=0; myEnum<32; myEnum++)
      {
            if(aValue & Math.pow(2, myEnum))
            {
                  myArray[31-myEnum] = "1";
            }
            else
            {
                  myArray[31-myEnum] = "0";
            }
      }

      return myArray.join("");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Binary bitwise operator,Bit-field, Bitwise AND then assign (&=), Bitwise expression, Bitwise operator, Logical AND (&&),Operator Precedence

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.10

Bitwise AND then assign (&=) (Operator/assignment)

Bitwise AND two operands and assign the result to the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 &= anOperand2`
Argument list:	`anOperand1`	A binary value
	`anOperand2`	Another binary value

Bitwise AND the right operand with the left operand and assign the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 & anOperand2;
```

Although this is classified as an assignment operator it is really a compound of an assignment and a bitwise operator.

The associativity is right to left.

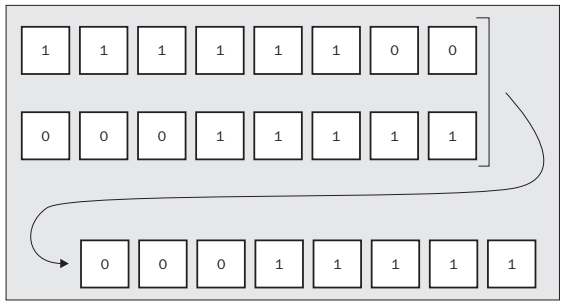
Refer to the Operator Precedence topic for details of execution order.

The new value of `anOperand1` is returned as a result of the expression.

The truth table shows the result of this operator for two Boolean primitive values:

A	B	AND
false	false	false
false	true	false
true	false	false
true	true	true

This is applied to each corresponding bit pair in the two values.



Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Assignment operator, Associativity, Bit-field,Bitwise AND (&), Bitwise expression, Bitwise operator, Logical AND (&&), LValue, Operator Precedence

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

Bitwise expression (Definition)

An expression that applies in a bitwise manner.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

Bitwise expressions perform a bit by bit operation across the entire integer width of the values.

See also:	Bit-field, Bitwise AND (&), Bitwise AND then assign (&=), Expression

Cross-references:

ECMA 262 edition 2 – section – 11.7

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.7

ECMA 262 edition 3 – section – 11.10

Bitwise NOT - complement (~) (Operator/bitwise)

Bitwise NOT of one operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

Property/method value type:	Number primitive	
JavaScript syntax:	-	~anOperand
Argument list:	anOperand	A numerical value

The operand is evaluated and then converted to a 32-bit integer value. Every bit is complemented and the result is a bitwise NOT.

The truth table shows the result of this operator for a Boolean primitive value:

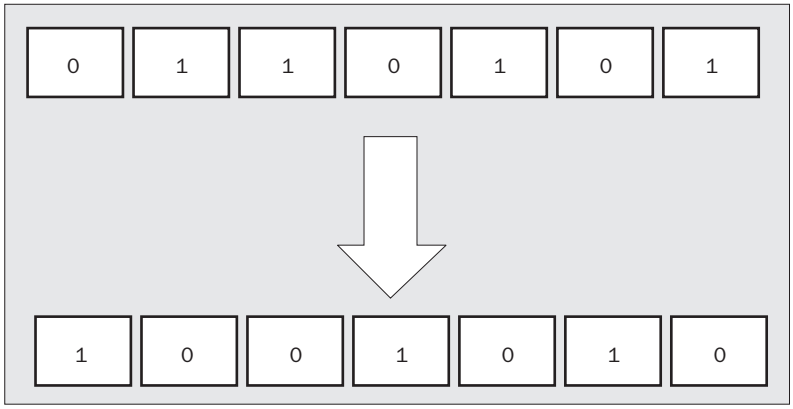
A	NOT
false	true
true	false

This operation is applied to each individual bit in the operand, inverting them one by one.

Note that this could be classified as a unary operator but here we have called it a bitwise operator on account of its functionality rather than its placement.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.



Warnings:

- There are some deficiencies in the handling of bitwise operators in the MSIE 5.0 browser on the Macintosh platform. It does not properly handle the sign bit and so you should observe some caution when using this operator.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myValue1 = 0xFFFF;
myValue2 = ~myValue1
document.write("Val 1 : " + binary32(myValue1) + "<BR>");
document.write("NOT : " + binary32(myValue2) + "<BR>");
// Binary convertor (ignore sign bit on MSIE)
function binary32(aValue)
{
  myArray = new Array(32);

  for(myEnum=0; myEnum<32; myEnum++)
  {
    if(aValue & Math.pow(2, myEnum))
    {
      myArray[31-myEnum] = "1";
    }
    else
    {
      myArray[31-myEnum] = "0";
  }
}
return myArray.join("");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Bit-field,Logical NOT – complement (!), Operator Precedence, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.4.8

ECMA 262 edition 3 – section – 11.4.8

Bitwise operator (Definition)

An operator that is applied in a bitwise manner.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

Bitwise operators convert both operands to 32 bit integers and apply the operator to them on a bit-by-bit basis.

Here is a table of all operators in the bitwise category and those are members of other categories but perform bitwise operations:

Op	Description
~	Bitwise complement (NOT)
&	Bitwise AND
<<	Bitwise left shift
>>	Bitwise right shift
>>>	Bitwise right shift (unsigned)
|	Bitwise inclusive OR
^	Bitwise XOR (exclusive OR)
&=	Bitwise AND and assign to an LValue
|=	Bitwise inclusive OR and assign to an LValue
^=	Bitwise exclusive XOR and assign to an LValue
<<=	Bitwise shift left and assign to an LValue
>>=	Bitwise shift right and assign to an LValue
>>>=	Bitwise shift right (unsigned) and assign to an LValue

Warnings:

- The result of a bitwise expression is a 32 bit binary value and should not be confused with the Boolean value returned by a logical operator.
- The bitwise operators may yield a value that in other languages is the same as the logical operator. However although in the C language, `true` and `false` are really integer values, in JavaScript the Boolean and Number values are distinctly different types.
- Be careful to use the correct number of ampersands and vertical bars to select the bitwise version of the operator. Refer to the Logical operator topic for a list of operators to avoid in bitwise expressions.

See also:	Associativity, Binary bitwise operator, Bit-field,Bitwise AND (&), Bitwise AND then assign (&=), Bitwise OR then assign (|=), Bitwise shift left then assign (<<=), Bitwise shift right and assign (>>=), Bitwise unsigned shift right and assign (>>>=), Bitwise XOR and assign (^=),Logical operator, Operator, Operator Precedence,Type conversion

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.10

Wrox *Instant JavaScript* – page – 19

Bitwise OR (|) (Operator/bitwise)

Bitwise OR of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 | anOperand2`
Argument list:	`anOperand1`	A numeric value
-	`anOperand2`	Another numeric value

Performs a bit-by-bit OR of the 32-bit value derived from both operands.

Where a corresponding bit is 1 in either of the two operands, a 1 is inserted into the result. A zero is inserted only when neither operand has a 1 bit at that position.

The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.

The truth table shows the result of this operator for two Boolean primitive values:

A	B	OR
false	false	false
false	true	true
true	false	true
true	true	true

This is applied to each corresponding bit pair in the operands.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myValue1 = 0x00FF;
myValue2 = 0xFF00;
myValue3 = myValue1 | myValue2;
document.write("Val 1 : " + binary32(myValue1) + "<BR>");
document.write("Val 2 : " + binary32(myValue2) + "<BR>");
```

```
document.write("OR : " + binary32(myValue3) + "<BR>");
// Binary convertor (ignore sign bit on MSIE)
function binary32(aValue)
{
  myArray = new Array(32);

  for(myEnum=0; myEnum<32; myEnum++)
  {
    if(aValue & Math.pow(2, myEnum))
    {
      myArray[31-myEnum] = "1";
    }
    else
    {
    myArray[31-myEnum] = "0";
  }
}
return myArray.join("");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Binary bitwise operator, Bit-field,Bitwise OR then assign (|=), Operator Precedence

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.10

Bitwise OR then assign (|=) (Operator/assignment)

Bitwise OR two operands and assign the result to the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 |= anOperand2`
Argument list:	`anOperand1`	A numeric value that can be assigned to
-	`anOperand2`	Another numeric value

Bitwise OR the right operand with the left operand and assign the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 | anOperand2;
```

Performs a bit by bit OR of the 32-bit value derived from both operands.

Where a corresponding bit is 1 in either of the two operands, a 1 is inserted into the result. A zero is inserted only when neither operand has a 1 bit at that position.

Although this is classified as an assignment operator it is really a compound of an assignment and a bitwise operator.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

The new value of `anOperand1` is returned as a result of the expression.

The truth table shows the result of this operator for two Boolean primitive values:

A	B	OR
false	false	false
false	true	true
true	false	true
true	true	true

This is applied to each corresponding bit pair in the operands.

Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Assignment operator, Associativity, Bit-field, Bitwise operator, Bitwise OR (|),LValue, Operator Precedence

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

Bitwise shift left (<<) (Operator/bitwise)

Bitwise shift leftwards one operand according to another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 << anOperand2`
Argument list:	`anOperand1` `anOperand2`	A value to be shifted A distance to shift `anOperand1`

The bitwise shift left operator converts its left operand to a 32 bit integer and moves it leftwards by the number of bits indicated by the right operand.

As the value is shifted leftwards, bits that roll out of the left end of the register are discarded. The right-hand end of the register is filled with zero bits. Shifting leftwards by 32 bits will fill the register with all zero bits.

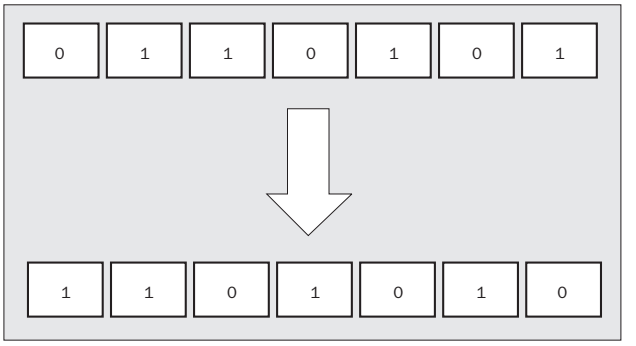
Because the value is converted to an integer, any fractional part is discarded as the shift begins.

The right-hand operand is converted to a 5 bit value with a bitwise mask to limit the distance of the shift to 32 bits. This can cause unexpected results if the right-hand side is derived from an expression that may yield a value larger than 32.

The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.

You can accomplish bitwise shift lefts by multiplying values using powers of 2. Multiplying a value by 2 shifts leftwards by one bit position.



Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myValue1 = 0x00FF;
myValue2 = myValue1 << 4;
document.write("Val 1 : " + binary32(myValue1) + "<BR>");
document.write("Result : " + binary32(myValue2) + "<BR>");
// Binary convertor (ignore sign bit on MSIE)
function binary32(aValue)
{
   myArray = new Array(32);

   for(myEnum=0; myEnum<32; myEnum++)
   {
      if(aValue & Math.pow(2, myEnum))
      {
         myArray[31-myEnum] = "1";
      }
      else
      {
         myArray[31-myEnum] = "0";
      }
   }
return myArray.join("");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Bit-field,Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), Operator Precedence, Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.7.1

ECMA 262 edition 3 – section – 11.7.1

Bitwise shift left then assign (<<=) (Operator/assignment)

Destructively bitwise leftwards shift the first of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera browser – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 <<= anOperand2`
Argument list:	`anOperand1`	A value to be shifted and assigned to
	`anOperand2`	A distance to shift `anOperand1`

Bitwise shift leftwards the left operand by the number of bits in the right operand and assign the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 << anOperand2;
```

The bitwise shift left operator converts it left operand to a 32 bit integer and moves it leftwards by the number of bits indicated by the right operand.

As the value is shifted leftwards, bits that roll out of the left end of the register are discarded. The right-hand end of the register is filled with zero bits. Shifting leftwards by 32 bits will fill the left operand with all zero bits.

Because the value is converted to an integer, any fractional part is discarded as the shift begins.

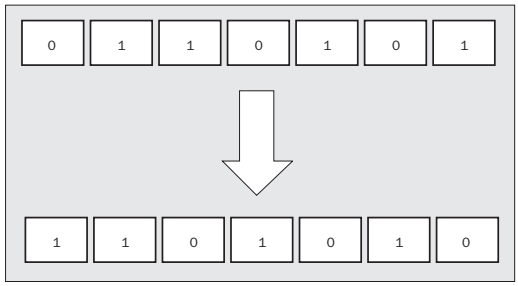
The right-hand operand is converted to a 5-bit value with a bitwise mask to limit the distance of the shift to 32 bits. This can cause unexpected results if the right-hand side is derived from an expression that may yield a value larger than 32.

Although this is classified as an assignment operator it is really a compound of an assignment and a bitwise operator.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

The new value of `anOperand1` is returned as a result of the expression.



Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Assignment operator, Associativity, Bit-field,Bitwise operator, Bitwise shift left (<<), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), LValue,Operator Precedence,Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

Bitwise shift operator (Definition)

A shift operator that moves the bits in an operand as if it were a shift register.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

Bitwise shift operators convert their left operands to a 32-bit integer value and shift them according to their right operand. The operator determines the kind of shifting that is applied.

See also:	Bit-field, Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=),Shift expression, Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.7

ECMA 262 edition 3 – section – 11.7

Bitwise shift right (>>) (Operator/bitwise)

Bitwise shift right one operand according to another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 >> anOperand2`
Argument list:	`anOperand1`	A value to be shifted
	`anOperand2`	A distance to shift the left operand

This is sometimes called shift right with sign extension.

The bitwise shift right operator converts its left operand to a 32 bit integer and moves it rightwards by the number of bits indicated by the right operand.

As the value is shifted rightwards, bits that roll out of the right end of the register are discarded. The left-hand end of the register containing the sign bit is duplicated to sign-fill the value as it shifts. Shifting rightwards by 32 bits will fill the register with all zero or all one bits according to the value of the sign bit at the outset.

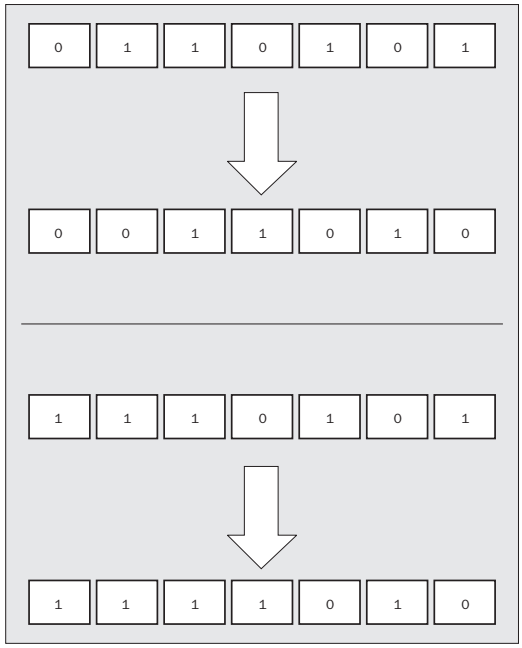
Because the value is converted to an integer, any fractional part is discarded as the shift begins.

The right-hand operand is converted to a 5-bit value with a bitwise mask to limit the distance of the shift to 32 bits. This can cause unexpected results if the right-hand side is derived from an expression that may yield a value larger than 32.

The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.

You can accomplish bitwise shift rights by dividing values using powers of 2. Dividing a value by 2 shifts rightwards by one bit position.



Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myValue1 = 0x00FF00;
myValue2 = myValue1 >> 4;
document.write("Val 1 : " + binary32(myValue1) + "<BR>");
document.write("Result : " + binary32(myValue2) + "<BR>");
// Binary convertor (ignore sign bit on MSIE)
function binary32(aValue)
{
   myArray = new Array(32);

   for(myEnum=0; myEnum<32; myEnum++)
   {
      if(aValue & Math.pow(2, myEnum))
      {
         myArray[31-myEnum] = "1";
      }
      else
      {
         myArray[31-myEnum] = "0";
      }
   }
   return myArray.join("");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Bit-field,Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), Operator Precedence, Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.7.2

ECMA 262 edition 3 – section – 11.7.2

Bitwise shift right and assign (>>=) (Operator/assignment)

Destructively bitwise rightwards shift the first of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 >>= anOperand2`
Argument list:	`anOperand1`	A value to be shifted and assigned to
	`="c2">anOperand2`	="c3">A distance to shift `anOperand1`

Bitwise shift rightwards the left operand by the number of bits in the right operand and assign the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 >> anOperand2;
```

The bitwise shift right operator converts its left operand to a 32 bit integer and moves it rightwards by the number of bits indicated by the right operand.

As the value is shifted rightwards, bits that roll out of the right end of the register are discarded. The left-hand end of the register containing the sign bit is duplicated to sign fill the value as it shifts. Shifting rightwards by 32 bits will fill the left operand with all zero or all one bits according to the value of the sign bit at the outset.

Because the value is converted to an integer, any fractional part is discarded as the shift begins.

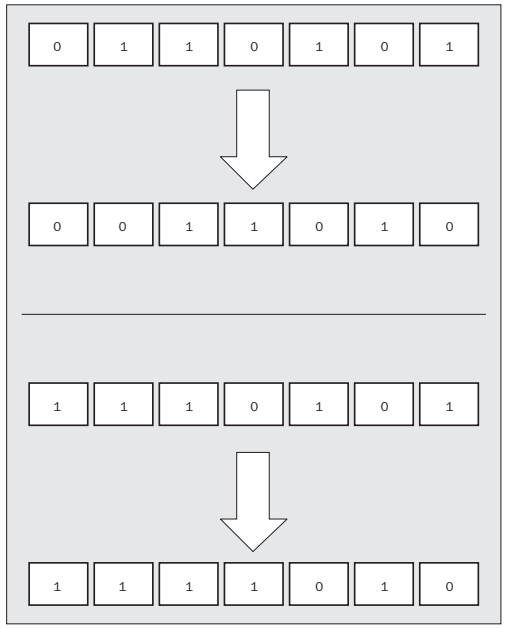
The right-hand operand is converted to a 5-bit value with a bitwise mask to limit the distance of the shift to 32 bits. This can cause unexpected results if the right-hand side is derived from an expression that may yield a value larger than 32.

Although this is classified as an assignment operator it is really a compound of an assignment and a bitwise operator.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

The new value of `anOperand1` is returned as a result of the expression.



Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Assignment operator, Associativity, Bit-field, Bitwise operator, Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=),LValue, Operator Precedence, Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

Bitwise unsigned shift right (>>>) (Operator/bitwise)

Bitwise shift right one operand according to another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 >>> anOperand2`
Argument list:	`anOperand1`	A value to be shifted
	`anOperand2`	A distance to shift the left operand

This is sometimes called shift right with zero extension.

The bitwise unsigned shift right operator converts its left operand to a 32 bit integer and moves it rightwards by the number of bits indicated by the right operand. The sign bit is not propagated.

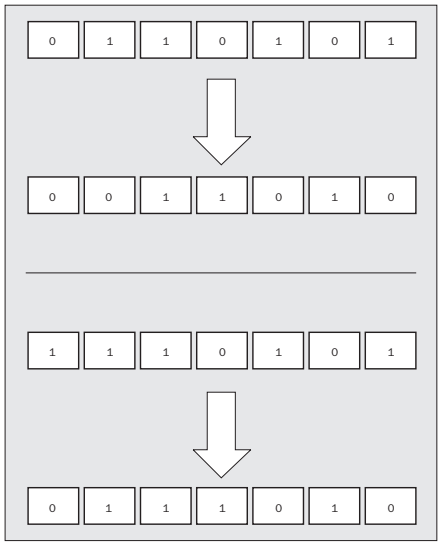
As the value is shifted rightwards, bits that roll out of the right end of the register are discarded. The left-hand end of the register containing the sign bit is zero-filled as the contents are shifted. Shifting rightwards by 32 bits will fill the register with all zero bits.

Because the value is converted to an integer, any fractional part is discarded as the shift begins.

The right-hand operand is converted to a 5-bit value with a bitwise mask to limit the distance of the shift to 32 bits. This can cause unexpected results if the right-hand side is derived from an expression that may yield a value larger than 32.

The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.



Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myValue1 = -0x00FF00;
myValue2 = myValue1 >>> 4;
document.write("Val 1 : " + binary32(myValue1) + "<BR>");
document.write("Result : " + binary32(myValue2) + "<BR>");
// Binary convertor (ignore sign bit on MSIE)
function binary32(aValue)
{
   myArray = new Array(32);

   for(myEnum=0; myEnum<32; myEnum++)
   {
      if(aValue & Math.pow(2, myEnum))
      {
         myArray[31-myEnum] = "1";
      }
      else
      {
         myArray[31-myEnum] = "0";
      }
   }
   return myArray.join("");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Bit-field,Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right and assign (>>>=), Operator Precedence, Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.7.3

ECMA 262 edition 3 – section – 11.7.3

Bitwise unsigned shift right and assign (>>>=) (Operator/assignment)

Destructively bitwise rightwards shift the first of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 >>>= anOperand2`
Argument list:	`anOperand1`	A value to be shifted and assigned to
	`anOperand2`	A distance to shift the left operand

Bitwise unsigned shift rightwards the left operand by the number of bits in the right operand and assign the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 >>> anOperand2;
```

The bitwise unsigned shift right operator converts its left operand to a 32 bit integer and moves it rightwards by the number of bits indicated by the right operand. The sign bit is not propagated.

As the value is shifted rightwards, bits that roll out of the right end of the register are discarded. The left-hand end of the register containing the sign bit is zero-filled as the contents are shifted. Shifting rightwards by 32 bits will fill the left operand with all zero bits.

Because the value is converted to an integer, any fractional part is discarded as the shift begins.

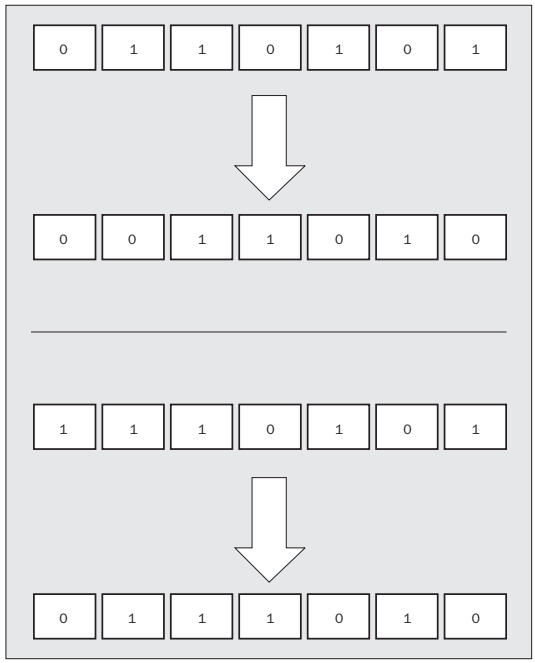
The right-hand operand is converted to a 5-bit value with a bitwise mask to limit the distance of the shift to 32 bits. This can cause unexpected results if the right-hand side is derived from an expression that may yield a value larger than 32.

Although this is classified as an assignment operator it is really a compound of an assignment and a bitwise operator.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

The new value of `anOperand1` is returned as a result of the expression.



Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Assignment operator, Associativity, Bit-field, Bitwise operator, Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), LValue, Operator Precedence, Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

Bitwise XOR (^) (Operator/bitwise)

Bitwise XOR one operand with another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 ^ anOperand2`
Argument list:	`anOperand1`	A numeric value
	`anOperand2`	Another numeric value

Performs a bit-by-bit XOR of the 32-bit values derived from both operands.

Where a corresponding bit is different in both operands, a 1 bit will be inserted into the result. If the corresponding bit is identical in both operands, regardless of whether they both have a 1 bit or a zero bit, a zero will be inserted at that bit position in the result.

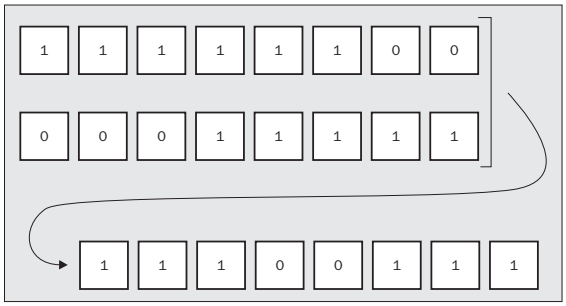
The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.

This is the truth table for two Boolean primitive values being operated on with the XOR operator.

A	B	XOR
false	false	false
false	true	true
true	false	true
true	true	false

The bitwise operator performs this operation on each corresponding bit pair in the two operands.



Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myValue1 = 0xFFFF;
myValue2 = 0x0FF0;
myValue3 = myValue1 ^ myValue2;
document.write("Val 1 : " + binary32(myValue1) + "<BR>");
document.write("Val 2 : " + binary32(myValue2) + "<BR>");
document.write("XOR : " + binary32(myValue3) + "<BR>");
// Binary convertor (ignore sign bit on MSIE)
function binary32(aValue)
{
   myArray = new Array(32);

   for(myEnum=0; myEnum<32; myEnum++)
   {
      if(aValue & Math.pow(2, myEnum))
      {
         myArray[31-myEnum] = "1";
      }
      else
      {
         myArray[31-myEnum] = "0";
      }
   }
   return myArray.join("");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Binary bitwise operator, Bit-field, Bitwise XOR and assign (^=),Logical XOR, Operator Precedence

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.10

Bitwise XOR and assign (^=) (Operator/assignment)

Destructively bitwise XOR two operands and store the result in the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

Property/method value type:	Number primitive	
JavaScript syntax:	-	anOperand1 ^= anOperand2
Argument list:	anOperand1	A numeric value that can be assigned to
	anOperand2	Another numeric value

Bitwise XOR the right operand with the left operand and assign the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 ^ anOperand2;
```

Performs a bit-by-bit XOR of the 32-bit values derived from both operands.

Where a corresponding bit is different in both operands, a 1 bit will be inserted into the result. If the corresponding bit is identical in both operands, regardless of whether they both have a 1 bit or a zero bit, a zero will be inserted at that bit position in the result.

Although this is classified as an assignment operator it is really a compound of an assignment and a bitwise operator.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

The new value of *anOperand1* is returned as a result of the expression.

This is the truth table for two Boolean primitive values being operated on with the XOR operator

A	B	XOR
false	false	false
false	true	true
true	false	true
true	true	false

The bitwise operator performs this operation on each corresponding bit pair in the two operands.

Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Assignment operator, Associativity, Bit-field, Bitwise operator, Bitwise XOR (^), LValue, Operator Precedence

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

BlendTrans() (Filter/blend)

A blend filter for controlling transitions.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

filter – `BlendTrans()`

Blinds() (Filter/transition)

A transition effect with the appearance of venetian blinds opening or closing.

Availability:	JScript – 5 Internet Explorer – 5

Refer to:

filter – `Blinds()`

blob object (Object/NES)

A special object that is designed to contain binary data extracted from a database or file.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	`myBlob = blob()`
	NES	`myBlob = myCursor.colName.blobImage(...)`
Object methods:	`<methodname>blobImage()</methodname>`, `<methodname>blobLink()</methodname>`	

A `blob` object is so called because it encapsulates a Binary Large OBject or BLOB. This is a block of data, often quite large, that is stored in a binary form and which is likely to contain many non-printable characters and probably some nulls as well.

You cannot instantiate a `blob` object directly in JavaScript but you can obtain one by fetching the data from a database as shown in the example code.

Example code:

```
<SERVER>
// Example derived from Wrox Professional JavaScript
// This opens a database, selects some records
// Traverses the collection that was selected
// and for each one, outputs an image tag.
database.connect("ODBC", "TargetDB", "", "", "");
myCursor = database.cursor("SELECT * FROM TARGET_TABLE");
while(myCursor.next())
{
   myBlob = myCursor.blobData;
   write(myBlob.blobImage("bmp"));
}
myCursor.close();
</SERVER>
```

See also:	Netscape Enterprise Server, `unwatch()`, `watch()`

Method	JavaScript	JScript	NES	Notes
`blobImage()`	1.1 +	-	2.0 +	-
`blobLink()`	1.1 +	-	2.0 +	-

blob.blobImage() (Method)

This method creates an `<IMG>` element having the appropriate MIME type for the `blob` object.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Image` object	
JavaScript syntax:	NES	`myBlob.blobImage(aFormat)`
	NES	`myBlob.blobImage(aFormat, aTxt)`
	NES	`myBlob.blobImage(aFormat, aTxt, anAlign)`
	NES	`myBlob.blobImage(aFormat, aTxt, anAlign, aPixWid)`
	NES	`myBlob.blobImage(aFormat, aTxt, anAlign, aPixWid, aPixHgt)`
	NES	`myBlob.blobImage(aFormat, aTxt, anAlign, aPixWid, aPixHgt, aPixBrdr)`
	NES	`myBlob.blobImage(aFormat, aTxt, anAlign, aPixWid, aPixHgt, aPixBrdr, isMap)`
Argument list:	`aFormat`	Image file format
	`anAlign`	The alignment of the image
	`aPixBrdr`	The border value
	`aPixHgt`	The height of the image
	`aPixWid`	The width of the image
	`aTxt`	The alt text for the image
	`isMap`	Whether the image is a map

The data is pulled out of the database according to the specified parameters. The BLOB can then be displayed as if it were an image in an <IMG> tag.

The format argument should contain an image specifier such as "GIF" or "JPEG" that can map conveniently to a file extension or MIME type.

The remaining parameters to this method mainly correspond to the HTML tag attributes that can be used with an <IMG> tag and are optional.

This method generates the necessary <IMG> tag to place into a document that refers to the BLOB data as if it were an image file on the server. When the document is parsed, the browser will request the image in the normal way; the contents of the BLOB are then returned in response to that request. The browser is not aware that the image data was retrieved from the database and by caching the image in memory when the link to it is placed in the document most of the latency associated with requesting objects out of the database is eliminated, albeit at the cost of increased memory usage in the server backend.

See also:	Cursor.blobImage(),blob object,MIME types

blob.blobLink() (Method)

This method creates an <A> element that links to the BLOB data.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Anchor` object	
JavaScript syntax:	NES	`myBlob.blobLink(aMimeType, aString)`
Argument list:	`aString`	The text inside the link
	`aMimeType`	The MIME type of the document being displayed

The data is pulled out of the database according to the specified parameters. The BLOB can then be displayed as if it were a document in an <A> tag.

This method generates the necessary URL to place into a document that links to it. If the user clicks on the link, the contents of the BLOB are then returned in response to that request.

See also:	`Cursor.blobLink(),blob` object, MIME types

Block { } (Statement)

A list of executable statements enclosed in curly braces.

Availability:	ECMAScript edition – 2
See also:	Compound statement, `if( ... ) ..., if( ... ) ... else ...`, Statement, Code block delimiter {}

Cross-references:

ECMA 262 edition 2 – section – 12.1

ECMA 262 edition 3 – section – 12.1

Wrox *Instant JavaScript* – page – 17

Block-level tag (Definition)

A block-level tag cannot exist inside a line. It must be placed on a line by itself.

By default, block-level items will be placed on a line by themselves because they force a line break before and after they are displayed. However you can modify the alignment and text flow around a block-level object to make it appear to be inline.

BLOCKQUOTE object (Object/HTML)

An object that represents a `<BLOCKQUOTE>` text area.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`myBLOCKQUOTE = myDocument.all.anElementID`
	IE	`myBLOCKQUOTE = myDocument.all.tags ("BLOCKQUOTE")[anIndex]`
	IE	`myBLOCKQUOTE = myDocument.all[aName]`
	-	`myBLOCKQUOTE = myDocument.getElementById (anElementID)`
	-	`myBLOCKQUOTE = myDocument.getElementsByName (aName)[anIndex]`
	-	`myBLOCKQUOTE = myDocument.getElementsByTagName("BLOCKQUOTE")[ anIndex]`

HTML syntax:	`<BLOCKQUOTE> ... </BLOCKQUOTE>`	
Argument list:	`anIndex`	A valid reference to an item in the collection
	`aName`	The `name` attribute of an element
	`anElementID`	The `ID` attribute of an element
Object properties:	`cite`	
Object methods:	`click()`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

This is used to set off a long quote inside a document and is intended to place an extract from a document into the displayed window with an active link to the document it quotes from. The style and appearance is that of a block quote text.

The `<BLOCKQUOTE>` tag is a block-level tag. That means that it forces a line break before and after unless the alignment and text flow around it are controlled very cleverly.

The DOM level 1 specification refers to this as a `QuoteElement` object.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`cite`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`click()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

BLOCKQUOTE.cite (Property)

A URL pointing at the document that a quote is attributed to.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBLOCKQUOTE.cite`

The URL of the document being quoted from is noted in this property.

Blur() (Filter/visual)

A visual filter for blurring objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Filter – `Blur()`

blur() (Method)

Move the input focus away from the receiving element.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 3.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`blur()`
	-	`myWindow.blur()`
See also:	`Input.blur(),Window.focus(),Window.blur()`	

BODY object (Object/HTML)

An object that represents the body of a document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`myBODY = myDocument.all.anElementID`
	IE	`myBODY = myDocument.all.tags("BODY") [anIndex]`
	IE	`myBODY = myDocument.all[aName]`
	-	`myBODY = myDocument.body`
	-	`myBODY = myDocument.getElementById (anElementID)`
	-	`myBODY = myDocument.getElementsByName (aName)[anIndex]`
	-	`myBODY = myDocument.getElementsByTagName ("BODY")[anIndex]`
HTML syntax:	`<BODY> ... </BODY>`	
Argument list:	`anIndex`	A valid reference to an item in the collection (should be 0)
	`aName`	The `name` attribute of an element
	`anElementID`	The `ID` attribute of an element
Object properties:	`accessKey, aLink,background,bgColor, bgProperties, bottomMargin, leftMargin, link,noWrap, recordNumber, rightMargin,scroll, tabIndex, text,topMargin, vLink`	
Object methods:	`createControlRange(), createTextRange()`	
Event handlers:	`onAfterUpdate, onBeforeUnload,onBeforeUpdate, onChange, onClick, onDataAvailable, onDataSetChanged, onDataSetComplete, onDblClick, onDragStart, onErrorUpdate, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onRowEnter, onRowExit, onScroll, onSelectStart, onUnload`	
Collections:	`controlRange[]`	

Although this generally represents the `<BODY>` tag, there are also properties that relate to the body that belong to the `Document` and `Window` objects. In MSIE, there is also a `HEAD` object, which contains related information.

The `<BODY>` tag is a block-level tag. You can't place a `<BODY>` tag into the document but taken in the context of a framed environment it manifests itself as if it were a block-level tag.

See also:	`Background` object, `Document` object, `Document.bgColor`, `Document.body`, `Element` object, `Element.isTextEdit`, `Element.offsetParent`, `Frame` object,`HEAD` object, `Input.accessKey`, `Window` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
aLink	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
background	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
bgColor	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
bgProperties	-	3.0 +	-	4.0 +	-	-	-	-
bottomMargin	-	3.0 +	-	4.0 +	-	-	-	Warning
leftMargin	-	3.0 +	-	4.0 +	-	-	-	Warning
link	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
noWrap	-	3.0 +	-	4.0 +	-	-	-	Warning
recordNumber	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
rightMargin	-	3.0 +	-	4.0 +	-	-	-	Warning
scroll	-	3.0 +	-	4.0 +	-	-	-	Warning
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
text	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
topMargin	-	3.0 +	-	4.0 +	-	-	-	Warning
vLink	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
createControlRange()	-	5.0 +	-	5.0 +	-	-	-	-
createTextRange()	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUnload	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onChange	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDataAvailable	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetChanged	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetComplete	-	3.0 +	-	4.0 +	-	-	-	-
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-
onUnload	1.5 +	3.0 +	6.0 +	3.02 +	-	-	-	Warning

Inheritance chain:

`Element` object, `Node` object

BODY.aLink (Property)

The colour of an active link in the current page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBODY.aLink`

This value controls the text of active links in the document body. You should use the normal color values to define the required color.

This property is equivalent to the `ALINK` attribute of the `<BODY>` HTML tag. This is the color that is used while the mouse is over the link and the button is held down by the user.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse.

Note also that its property name is not consistent with its counterpart, the `document.alinkColor` property whose value it reflects.

See also:	BODY.link, BODY.text, BODY.vLink, Color names, Color value, Document.alinkColor, Document.bgColor, Document.fgColor, Document.linkColor, Document.vlinkColor, HTML object

BODY.background (Property)

The URL of a background image for the current document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBODY.background`

If a background image is available, then its URL is contained in this property. Changing the value in this property will replace the background with a new one; however, there may be a perceptible delay while the new image is fetched from the web server.

The background image for the document that is defined in the `<BODY>` tag is accessible via the `BODY.background` property in MSIE.

Warnings:

- You cannot access the background image directly in Netscape Navigator because the `BODY` object is not reflected into the JavaScript environment.

See also:	Background.src, BODY.bgProperties, Document.background, HTML object

BODY.bgColor (Property)

The background color of the current page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBODY.bgColor`

This corresponds to the `BGCOLOR=". . ."` HTML tag attribute on the `<BODY>` tag.

You can modify this value at any time, the result of which will be to change the background color of the page.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse. On the other hand it does work consistently on both MSIE and Netscape whereas style sheet controls do not. Later, a DOM-standardized approach to style handling will provide a cross-platform way to access style information from JavaScript. DOM level 2 introduces a first attempt at standardizing this area.

The background can be colored whether an image is loaded into the background of a document or not. In fact, it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

The background color for the document that is defined in the `<BODY>` tag is also reflected in the `bgColor` property of the `document` object, although that is now deprecated as of DOM level 1.

See also:	Color names, Color value, Document.bgColor, Document.linkColor ,`HTML` object

BODY.bgProperties (Property)

An attribute that controls the way the background image is managed when the page scrolls.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myBODY.bgProperties`

The value of this property can be set to either the value "`fixed`" or an empty string (""). When the "`fixed`" value is used, the background image (if it is specified) will be locked into position and if the page is scrolled, the background will stay fixed where it is as if it were on a separate layer.

See also:	Background.src, BODY.background

BODY.bottomMargin (Property)

A margin space at the bottom of the document in the current window.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myBODY`*`.bottomMargin`

Normally, you would not need to specify this property. It allows the document to have some clear space at the bottom so, for example, the content could be made to scroll completely off the screen.

The distance is measured in pixels and can range from zero (which is the default) to any reasonably sensible value.

If a page is being created dynamically with `document.write()` methods, and a script error occurs, the margin is not appended. It appears to be added as a property of the body closure.

This corresponds to the `style.marginBottom` property and the `margin-bottom` attribute that is defined in a style sheet.

Warnings:

- Even when this value is set to zero, the Macintosh version of MSIE has a noticeable margin at the bottom.

See also:	`BODY.topMargin`

BODY.controlRange[] (Collection)

A collection of all the elements within the document body.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	`myBODY.controlRange`

This collection is returned by the `createControlRange()` method. The items in this collection would all represent component elements within the page but would not include simple text items.

See also:	`BODY.createControlRange()`

Property attributes:

`ReadOnly`.

BODY.createControlRange() (Method)

A constructor function to create a new `controlRange` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`ControlRange` object	
JavaScript syntax:	IE	`myElement.createControlRange()`

This method creates a collection of non-text elements. These are selectable items based on controls rather than text. If the `controlRange` (belonging to the `BODY` object) already exists, then it will be overwritten by the results of this method call.

See also:	`BODY.controlRange[]`

BODY.createTextRange() (Method)

Used in MSIE for creating a text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`TextRange` object	
JavaScript syntax:	IE	`myBODY.createTextRange()`

This method should only be used if the receiving object responds `true` to its `isTextEdit` property request.

See also:	`TextRange` object

BODY.leftMargin (Property)

A margin down the left edge of the document window.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`myBODY.leftMargin`

This property controls the amount of space down the left margin of a page. This indents all of the content away from the left edge of its containing window or frame.

This corresponds to the `style.marginLeft` property and `margin-left` stylesheet attribute.

Warnings:

- Note that the default values are platform-dependent and although it is only a couple of pixels difference it can throw off the layout of a page significantly if you make the wrong assumption.

See also:	`BODY.rightMargin`

BODY.link (Property)

The color of an as yet unvisited link in the page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBODY.link`

This value controls the text of active links in the document body. You should use the normal color values to define the required color.

This property is equivalent to the `LINK` attribute of the `<BODY>` HTML tag. This is the color that is used for as yet unvisited links.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse.

Note also that its property name is not consistent with its counterpart, the `document.linkColor` property whose value it reflects.

See also:	`BODY.aLink`,`BODY. text`,`BODY.vLink`, Color names,Color value, `Document.alinkColor`, `Document.bgColor`, `Document.fgColor`, `Document.linkColor`, `Document.vlinkColor`, `HTML` object

BODY.noWrap (Property)

A switch to control whether text should wrap or not within the page.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`myBODY.noWrap`

This is a Boolean value that controls whether the textual content is wrapped at the right-hand window border or not.

If the value `false` is assigned to this property, then words will wrap as the page is drawn. This is the way you would expect a browser to behave. The text will flow according to the space available.

If the value `true` is assigned to this property, the line of text will continue to the right until a `<BR>` or other block level tag is encountered. This will force the horizontal width of the page to extremely large and the user will need to scroll furiously to be able to see the text and then scroll back again for the start of the next line.

Warnings:

- Only use this if you plan to place line breaks at frequent intervals yourself and really do need to control the line breaks manually.

BODY.recordNumber (Property)

The record within the dataset that defined the page content when the content came from a data source.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`myBODY.recordNumber`

This is a property that is part of the MSIE data-binding support. It contains an integer value that is the record number within the data set that created this object.

This is useful when you are building pages with ASP and Active Data Objects (ADO).

See also:	Active Server Pages, ADO

Property attributes:

`ReadOnly`.

BODY.rightMargin (Property)

A margin space down the right hand side of the page.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`myBODY.rightMargin`

This property controls the amount of space down the right-hand margin of a page. This indents all of the content away from the right edge of its containing window or frame.

This corresponds to the `style.marginRight` property and `margin-right` stylesheet attribute.

Warnings:

- Note that the default values are platform-dependent and, although it is only a couple of pixels difference, it can throw off the layout of a page significantly if you make the wrong assumption.

See also:	BODY.leftMargin

BODY.scroll (Property)

A switch for whether the scrollbars appear or not.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myBODY.scroll`

If this property contains "`yes`", then scrollbars will appear if the window content exceeds the size of the space available. If it is "`no`", then the scroll bars will not appear.

Although this is a switch, it is not strictly a Boolean value because it only takes the values "yes" and "no". A true Boolean value would accept only "true" or "false". This may be because the property might yield the value "auto" on some platform variants if it has been defined in the HTML tag attributes for a frame.

Warnings:

- Although the `BODY` object is not supported on Netscape Navigator, the HTML tag attribute properties that control scrollbar visibility are the same on both MSIE and Netscape Navigator.
- You should note, however, that the content of a page in Netscape Navigator cannot be scrolled unless the scrollbars are visible. Even if the content does not exceed the space available, the scroll bars will still be drawn but will be inactive. To scroll content in Netscape Navigator without scrollbars being visible, you will need to create a `<LAYER>` and scroll that.
- This leads to a further complication in that the vertical scroll value moves the content in the opposite direction in MSIE and Netscape Navigator as it is incremented.

BODY.tabIndex (Property)

An integer that represents the position of this document in the tabbing order.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`myBODY.tabIndex`

This value indicates where in the tabbing sequence this object and any of its children will be placed. The tabbing order is used when filling in forms. Pressing the [*tab*] key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

BODY.text (Property)

The color of body text within the page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBODY.text`

This value controls the foreground text in the document body. You should use the normal color values to define the required color.

This is the default text color for the document. It corresponds to the `TEXT` attribute in the `<BODY>` tag.

Default foreground text is colored according to this setting unless it is in an `<A>` tag when the `alinkColor`, `linkColor` and `vlinkColor` properties override it. The foreground text color can be changed inline with the `<FONT COLOR=". . .">` HTML tag attribute.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse.

Note also that its property name is not consistent with its counterpart, the `document.fgColor` property whose value it reflects.

See also:	`BODY.aLink`, `BODY.link`, `BODY.vLink`, Color names, Color value, `Document.alinkColor`, `Document.bgColor`, `Document.fgColor`, `Document.linkColor`, `Document.vlinkColor`, `HTML` object

BODY.topMargin (Property)

A margin value at the top of the document window.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myBODY*`.topMargin`

Normally, you would not need to specify this property. It allows the document to have some clear space at the top so for example the content could be made to scroll completely off the screen when it is loaded.

The distance is measured in pixels and can range from zero (which is the default) to any reasonably sensible value. Making it any bigger than the screen size is pointless.

This corresponds to the `style.marginTop` property and the `margin-top` attribute that is defined in a style sheet.

Warnings:

- Note that the default values are platform-dependant and although it is only a couple of pixels difference it can throw off the layout of a page significantly if you make the wrong assumption.

See also:	`BODY.bottomMargin`

BODY.vLink (Property)

The color of visited links within the page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBODY.vLink`

This value controls the text of visited links in the document body. You should use the normal color values to define the required color.

This corresponds to the `VLINK` attribute in the `<BODY>` tag.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse.

Note also that its property name is not consistent with its counterpart, the `document.vlinkColor` property whose value it reflects.

See also:	BODY.aLink, BODY.link, BODY.text, Document.alinkColor, Document.bgColor, Document.fgColor, Document.linkColor, Document.vlinkColor, HTML object

Bookmarklets (Advice)

A means of storing fragments of JavaScript for execution as bookmarks.

Creating a `javascript:` URL with some attached JavaScript code and storing it in the bookmarks or favorites of your browser is a way of setting up some really useful debugging tools. It seems to work in most browsers, although some have a size limit on the URL that you can use.

See also:	JavaScript Bookmark URLs, JavaScript interactive URL, `javascript:` URL

Boolean (Primitive value)

A built-in primitive value.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

A Boolean value is a member of the Boolean type and may have one of two unique values, either `true` or `false`.

In some languages the values `true` and `false` also equate to numeric values. False is commonly 0 and `true` any non-zero value. In JavaScript this is not the case. The value `false` does not test equal against zero. However, a `false` Boolean value does become zero when converted to a number.

If you create a `Boolean` object and set it to the value `true`, you cannot convert it to a number with the `toNumber()` method, because this generates a run-time error. However, you can coerce the Boolean value into a numeric value by preceding it with a unary plus sign. So `+true` is a numeric primitive and yields the value 1, while `false` is converted to zero.

See also:	`false`, JavaScript to Java values, `true`

Cross-references:

ECMA 262 edition 2 – section – 4.3.13

ECMA 262 edition 3 – section – 4.3.13

Wrox *Instant JavaScript* – page – 14

boolean (Reserved word)

Reserved for future language enhancements.

The `boolean` keyword represents both a Java data type and the native Boolean primitive data type in JavaScript. This suggests some potential extensions of JavaScript interfaces to access Java applet parameters and return values.

See also:	`java.lang.Boolean`, `LiveConnect`, Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Boolean (Type)

A native built-in type.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

Any object or expression that yields a result of type Boolean represents a logical entity.

Logical entities can only represent the `true` or `false` states.

These are useful as flags or conditional switches in your script.

See also:	Data Type, `false`, Fundamental data type, `true`, Type

Cross-references:

ECMA 262 edition 2 – section – 4.3.14

ECMA 262 edition 2 – section – 8.3

ECMA 262 edition 3 – section – 4.3.14

ECMA 262 edition 3 – section – 8.3

O'Reilly *JavaScript Definitive Guide* – page – 41

Boolean literal (Primitive value)

A literal constant whose type is a built-in primitive value.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

Boolean literals specify constant values for the `true` and `false` values used in relational expressions and are the only two values a Boolean primitive or object can resolve to.

See also:	`false`, Implicit conversion, Literal, Token, `true`

Cross-references:

ECMA 262 edition 2 – section – 7.7.2

ECMA 262 edition 3 – section – 7.8.2

Boolean object (Object/core)

An object of the class "`Boolean`".

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	`myBoolean = BooleanValue`
	-	`myBoolean = new Boolean()`
	-	`myBoolean = new Boolean(aValue)`
Argument List	`BooleanValue`	`A Boolean value (either true or false)`
	`aValue`	A value to be converted to a `Boolean` object.
Object properties:	`constructor,prototype`	
Object methods:	`toSource(),toString(),valueOf()`	

An instance of the class "`Boolean`" is created by using the `new` operator on the `Boolean()` constructor. The new object adopts the behavior of the built-in `Boolean` prototype object through the prototype-inheritance mechanisms.

All properties and methods of the prototype are available as if they were part of the new instance.

A `Boolean` object is a member of the type Object and is an instance of the built-in `Boolean` object.

Cloning the built-in `Boolean` object creates `Boolean` objects. This is done by calling the `Boolean()` constructor with the `new` operator. For example:

```
myBoolean = new Boolean(true);
```

A `Boolean` object can be coerced into a Boolean value and can be used anywhere that a Boolean value would be expected.

Programmers familiar with object-oriented techniques may be happy to use the `Boolean` object, while procedural language programmers may prefer to implement the same functionality with a Boolean value instead.

This is an example of the flexibility of JavaScript in its ability to accommodate a variety of users from different backgrounds.

The prototype for the `Boolean` prototype object is the `Object` prototype object.

See also:	`Boolean. prototype`, Native object, `Object` object, `unwatch()`, `watch()`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`constructor`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	2 +	-
`prototype`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	-

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`toSource()`	1.3 +	-	4.06 +	-	3.0 +	-	-	-
`toString()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	-
`valueOf()`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	2 +	-

Cross-references:

ECMA 262 edition 2 – section – 4.3.15

ECMA 262 edition 2 – section – 10.1.5

ECMA 262 edition 2 – section – 15.6

ECMA 262 edition 3 – section – 4.3.15

ECMA 262 edition 3 – section – 15.6

Boolean() (Constructor)

A Boolean object constructor.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Boolean` object	
JavaScript syntax:	-	`new Boolean()`
	-	`new Boolean(aValue)`
Argument list:	`aValue`	A value to be converted to a `Boolean` object.

The `Boolean()` constructor is used to manufacture new instances of the built-in `Boolean` object.

When the `Boolean()` constructor is called by the `new` operator, it initializes a brand new `Boolean` object instance.

The value of the new `Boolean` object instance is the same as the Boolean value yielded by the type conversion of the `Boolean()` constructor's parameter.

Value:	Result:
No value	Always `false`
undefined	Always `false`
null	Always `false`
Boolean	No conversion, the input value is returned unchanged
Number	The result is `false` if the argument is 0 or NaN, otherwise it is true
String	Zero length strings return `false` otherwise the result is `true`
Object	Always `true`

The result of calling the constructor is a `Boolean` object whose value is `true` or `false` depending on the input value. If the value-input parameter is omitted, then a `Boolean` object with value `false` is returned by default.

Warnings:

- Note that unlike the `Object()` constructor, which can be called without its parentheses, calling the `Boolean()` constructor without parentheses yields an uninitialized object.
- Note also that using `Boolean` objects in conditional code is prone to risks due to the fact that all objects yield a Boolean `true` value when tested in logical expressions. This includes `Boolean` objects whose present value is `false`.

See also:	`Constructor` function, `constructor` property, `Global` object,`new`, Object constant, `Object()`

Cross-references:

ECMA 262 edition 2 – section – 15.1.3.5

ECMA 262 edition 2 – section – 15.6.1

ECMA 262 edition 2 – section – 15.6.2

ECMA 262 edition 2 – section – 15.6.3

ECMA 262 edition 2 – section – 15.6.3.1

ECMA 262 edition 3 – section – 15.6.2

Boolean() (Function)

A `Boolean` object constructor.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`Boolean()`
	-	`Boolean(aValue)`
Argument list:	`aValue`	A value to be converted to a Boolean result

When the `Boolean()` constructor is called as a function, it performs a type conversion on the value that is passed to it as a parameter.

The following results are yielded by the `Boolean()` constructor function:

Value:	Result:
No value	`false`
undefined	`false`
null	`false`
Boolean `false`	`false`
Boolean `true`	`true`
NAN	`false`
0	`false`
Non zero number	`true`
Zero length string ""	`false`
Non zero length string	`true`
Object	`true`

The result will be `true` or `false` depending on the parameter's value. If the parameter value is omitted, then `false` is returned by default.

See also:	`Cast` operator, `Constructor` function, `constructor` property, Implicit conversion, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 15.1.3.5

ECMA 262 edition 2 – section – 15.6.1

ECMA 262 edition 2 – section – 15.6.2

ECMA 262 edition 2 – section – 15.6.3

ECMA 262 edition 3 – section – 15.6.1

Boolean.Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition – 2

This is an internal property that describes the class that an instance of a `Boolean` object is a member of. The reserved words suggest that this property may be externalized in the future.

See also:	`Boolean.constructor`, Class

Property attributes:

`DontEnum`, `Internal`.

Cross-references:

ECMA 262 edition 2 – section – 15.6.4

Boolean.constructor (Property)

A reference to the constructor for the boolean object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Boolean constructor`	
JavaScript syntax:	-	`myBoolean.constructor`

The constructor referenced by this property is that of the built-in `Boolean` prototype object.

You can use this referenced constructor as one way of creating `Boolean` objects, although it is more popular to use the `new Boolean()` technique. This property is especially useful if you have an object that you want to clone, but you don't know what sort of object it is. Simply use the property to access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective, so there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	`Boolean.Class`, `Boolean.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.6.1

ECMA 262 edition 2 – section – 15.6.2

ECMA 262 edition 2 – section – 15.6.3

ECMA 262 edition 3 – section – 15.6.2

Boolean.prototype (Property)

The prototype for the `Boolean` object that can be used to extend the interface for all `Boolean` objects.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Boolean` object	
JavaScript syntax:	-	`Boolean.prototype`
	-	`myBoolean.constructor.prototype`

The initial value of the prototype of a `Boolean` object is the built-in `Boolean` prototype object.

The example demonstrates how to provide extensions to all instances of this class by adding a function to the prototype object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that extends the output capabilities of Boolean objects
function yesNo()
{
if(this == true)
{
return "The switch is ON";
}
else
{
return "The switch is OFF";
}
}
// Register the new function
Boolean.prototype.yesNo = yesNo;
```

```
// Create a Boolean object and test the Boolean.yesNo() method
myBoolean = new Boolean(true);
document.write(myBoolean.yesNo())
document.write("<BR>")
myBoolean = !myBoolean;
document.write(myBoolean.yesNo())
document.write("<BR>")
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Boolean` object,`Boolean.constructor`,`Boolean.toSource()`, `Boolean.toString()`,`Boolean.valueOf()`,`prototype` property, Boolean.prototype

Cross-references:

ECMA 262 edition 2 – section – 15.2.3.1

ECMA 262 edition 2 – section – 15.6.3.1

ECMA 262 edition 3 – section – 15.6.3.1

Boolean.toSource() (Method)

Returns a `Boolean` object formatted as a Boolean literal contained in a string.

Availability:	JavaScript – 1.3 Netscape – 4.06 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	`myBoolean.toSource()`

This is an alternative way to retrieve a string version of a Boolean value. In this case, it is formatted as a Boolean literal and can then be used in an `eval()` function to assign another Boolean.

If you run the example below, it should yield this as a result:

```
(new Boolean(true))
```

However, you should note that this is not supported by MSIE browsers.

The result of calling this method is a string version of the Boolean formatted as a Boolean literal.

Example code:

```
// Create a boolean and then examine its source
myBoolean = new Boolean(true);
document.write(myBoolean.toSource());
```

See also:	`Boolean.prototype`,`Boolean.toString()`

Boolean.toString() (Method)

Returns a string primitive version of an object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBoolean.toString()`

The value of the object is converted to a string that represents its Boolean value.

See also:	`Boolean.prototype`, `Boolean.toSource()`, `Cast` operator, `toString()`

Cross-references:

ECMA 262 edition 2 – section – 15.6.4.2

ECMA 262 edition 3 – section – 15.6.4.2

Boolean.valueOf() (Method)

Returns the primitive value of the Boolean object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`myBoolean.valueOf()`

The `Boolean` object is converted to a Boolean primitive and returned to the caller.

See also:	`Boolean.prototype`, `Cast` operator, `valueOf()`

Cross-references:

ECMA 262 edition 2 – section – 15.6.4.3

ECMA 262 edition 3 – section – 15.6.4.3

BR object (Object/HTML)

An object that represents the `<BR>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	`myBR = myDocument.all.anElementID`
	IE	`myBR = myDocument.all.tags("BR") [anIndex]`
	IE	`myBR = myDocument.all[aName]`
	-	`myBR = myDocument.getElementById (anElementID)`
	-	`myBR = myDocument.getElementsByName (aName)[anIndex]`
	-	`myBR = myDocument.getElementsByTagName ("BR")[anIndex]`
HTML syntax:	` `	
Argument list:	`anIndex`	A valid index reference to an item in the collection
	`aName`	The `name` attribute of an element
	`anElementID`	The `ID` attribute of an element
Object properties:	`clear`	
Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`	

This object represents a line break in the text. There are very few appearance-modifying properties you could apply to such an object.

The `<BR>` tag is a block-level tag. That means that it forces a line break before and after itself.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`clear`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

BR.clear (Property)

An property that controls how the browser treats the following paragraph alignment.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBR.clear`

The value of this property controls the way that text flows around inline images and other block-level objects. How this works will depend on how the object is fixed to either the left or right page border.

This property can contain any of the following values, or may simply contain an empty string:

- `all`
- `left`
- `right`

Additional accessor methods are defined in the DOM standard, but are yet to be implemented. Since this property is read/write accessible, they are largely unnecessary.

Braces { } (Delimiter)

A delimiting token for a block of executable script.

Availability:	ECMAScript edition – 2
See also:	`if( ... ) ...`,`if( ... ) ... else ...`, Code block delimiter {}

Cross-references:

ECMA 262 edition 2 – section – 12.5

ECMA 262 edition 3 – section – 12.1

break (Statement)

Exit unconditionally from a loop or switch.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	`break aLabelName;`
	-	`break;`
Argument list:	`aLabelName`	The name of a label associated with some code

The `break` keyword is a 'jump' statement. It is used in an loop to abort the current cycle and exit from the smallest enclosing loop immediately. Execution continues at the line following the statement block associated with the loop.

A break statement can only legally exist inside a `while` or `for` loop in an ECMA-compliant implementation. Implementations that provide additional iterator types may also honor the same behavior for the `break` statement.

The `break` statement would normally be executed conditionally, otherwise it would cause the remaining lines in the loop to be redundant, since no execution flow would ever reach them. Compilers generally warn you about this, but JavaScript would simply ignore it.

At version 1.2 of JavaScript, the `break` statement was enhanced to support a label as a breaking destination. When the break is processed, it will jump to the end of the statement that has been labeled. If an iterator is labeled, then the break is associated with that iterator. This mechanism works like a 'goto'. It can work with an `if` block and with a labeled block of brace delimited code.

See also:	Completion type,`continue`,`for( ... ) ...`,`for( ... in ... ) ...`, Iteration statement,Jump statement, Label,`return`, Scope chain,Statement, `switch( ... ) ... case: ... default: ...`, `while( ... ) ...`

Cross-references:

ECMA 262 edition 2 – section – 10.1.4

ECMA 262 edition 2 – section – 12.8

ECMA 262 edition 3 – section – 10.1.4

ECMA 262 edition 3 – section – 12.8

Wrox *Instant JavaScript* – page – 25

Broken-down time (Definition)

Time disassembled into component parts, for example hours or minutes.

Broken-down time is handled with a time structure in the C language. Breaking down a time value in that context can require several lines of code. However, JavaScript provides the `Date` object class, which supports methods for accessing the separate time and date components.

The following time components are available:

- Year number
- Month in year
- Date within month
- Day of week
- Hour within day
- Minute within hour
- Second within minute
- Millisecond within second
- Time of day in milliseconds

These values are available measured in either local time or UTC time.

There are also facilities to convert back and forth between local time and UTC time.

See also:	Calendar time, Date and time, Date from time, Date number, `Date` object, Day from year, Day number,Day within year, Daylight savings time adjustment, Days in year, Local time, Local time zone adjustment, Locale-specific behavior, Time from year, Time range, Time value, Time within day,`TimeClip()`, Universal coordinated time, Year from time, Year number

Browser (Object model)

The collection of objects that a browser manages.

Refer to:

Document

Browser detection (Advice)

Browser detection techniques require some review in the light of recent browser upgrades.

Browser-detection techniques have been in use ever since the MSIE version 3 browser was launched. Since the features of this browser differed from others, it became necessary to determine what browser was being used to establish the features actually available.

The subject of browser detection is now extremely complex, with a wide variety of browsers. The technique of simply distinguishing between Netscape and MSIE is no longer sufficient. This is even more of an issue now that Netscape version 6.0 is shipping.

This Netscape version 6.0 browser was developed around a completely new source code base. Having started completely afresh, Netscape has not implemented any unnecessary legacy features and instead has pursued a strictly standards based approach. The most noticeable effect of this is the complete lack of support for layers.

The reason why this is such a big problem is that in the past we might typically have written a short script to determine whether we were running Netscape or MSIE, and coded accordingly. The first example shows how we would have done this using a classical detection technique.

However, now we have millions of web pages using this technique that also use layers. They will detect Netscape 6.0 and return a value saying "OK it's Netscape, so use the Netscape layers code alternative". This is going to break a lot of pages.

We now need something that tells us whether a particular feature is available rather than a particular browser. The second example is a skeleton of how we might do that. It adds a member object called `isAvailable` to the global object, that can have additional properties added as we need to extend it. In this example, it simply provides access to whether layers are available or not. It provides for three possible cases:

- `NO` – Layers are not available and no alternative simulation is possible
- `DIV` – Layers are not available, use `<DIV>` blocks and CSS positioning
- `YES` – Layers are available

So now we can build some code to exploit this using a `switch` statement thus:

```
isAvailable();
switch(isAvailable.Layers)
{
   case "YES" :
// Call the layer programmed version of our page
      break;
   case "DIV" :
// Call the <DIV> simulated layers version of the page
```

```
      break;
   case "NO" :
// Fall back to the legacy browser, no layers page
      break;
   default:
// Unexpected condition handled here
      break;
}
```

Because we need to know what browser and version the code is being run on to determine simulation capabilities, those get set as member properties of the `isAvailable` object.

Later you could add capabilities to access the DOM feature-detection mechanisms so that all these related feature-detection facilities are in a single reusable code block.

Note that the `appVersion` value picks up the Mozilla/X.YY value and uses that. This means that MSIE 5 reports an `appVersion` of 4 and Netscape version 6.0 reports an `appVersion` of 5, which is odd to say the least. You can do a bit more work to parse out the correct version numbers from the remainder of the user agent string or access special values that are platform dependant within a fragment of code that is selected on a per platform basis.

Example code:

```
<SCRIPT>
// Classic browser detection returning browser types
// and versions
function getBrowserType()
{
var myUserAgent;
var myMajor;
myUserAgent   = navigator.userAgent.toLowerCase();
myMajor       = parseInt(navigator.appVersion);
if( (myUserAgent.indexOf('mozilla')    != -1) &&
(myUserAgent.indexOf('spoofer')    == -1) &&
(myUserAgent.indexOf('compatible') == -1) &&
(myUserAgent.indexOf('opera')      == -1) &&
(myUserAgent.indexOf('webtv')      == -1)
)
{
if (myMajor > 4)
{
return "nav6";
}
if (myMajor > 3)
{
return "nav4";
}
return "nav";
}
if (myUserAgent.indexOf("msie") != -1)
{
if (myMajor > 4)
{
return "ie5";
}
if (myMajor > 3)
{
return "ie4";
}
return "ie";
}
return "other";
}
```

```
</SCRIPT>


<SCRIPT>
isAvailable();
document.write("Browser family : "  + isAvailable.Browser + "<BR>");
document.write("Browser version : " + isAvailable.Version + "<BR>");
document.write("Layer support : "   + isAvailable.Layers  + "<BR>");
// Modern extensible browser feature detection
// Becomes a member property of the global object
function isAvailable()
{
// Get user agent stuff
var myUserAgent   = navigator.userAgent.toLowerCase();


// Set initial conditions
isAvailable.Browser = "OTHER";
// Check for navigator
if( (myUserAgent.indexOf('mozilla')    != -1) &&
(myUserAgent.indexOf('spoofer')    == -1) &&
(myUserAgent.indexOf('compatible') == -1) &&
(myUserAgent.indexOf('opera')      == -1) &&
(myUserAgent.indexOf('webtv')      == -1)
)
{
isAvailable.Browser = "NAV";
}
// Check for MSIE
if (myUserAgent.indexOf("msie") != -1)
{
isAvailable.Browser = "MSIE";
}
// Store major version number from leading Moziilla/X.YY
// Portion of user agent string
isAvailable.Version = parseInt(navigator.appVersion);

// Work out if layers available
if(document.layers)
{
isAvailable.Layers = "YES";
}
else
{
if((isAvailable.Browser = "MSIE") && (isAvailable.Version > 3) ||
(isAvailable.Browser = "NAV")  && (isAvailable.Version > 4))
{
isAvailable.Layers = "DIV";
}
else
{
isAvailable.Layers = "NO";
}
}
}
// For further investigation look at the MSIE script engine version and build
number properties and map them to features.
</SCRIPT>
```

Browser version compatibility (Advice)

Browser upgrades are not always upwardly compatible.

It is fairly obvious that as browsers are improved, new features will be added. This suggests that you might upgrade and begin to exploit those new features. At the next browser upgrade, these features should still be available, while yet more are introduced. This is called upwards compatibility. This is generally no problem.

Downwards compatibility, where code using features in a later browser does not cause errors in an earlier browser, is a little more difficult to provide. HTML has good downwards compatibility due mainly to the fact that if a tag is unrecognized, it is simply ignored. That means web pages containing new features simply display any contained text inside the unrecognized block as if the unsupported tag did not exist.

This may be easy to manage with HTML, but is not feasible with a scripting language because you can't expect the browser simply not to execute a line of script. However, you can code defensively in such a way that your scripts may be downwards compatible.

To code defensively means to check for the existence of a feature before using it, and also to check that objects are defined before trying to modify their properties. You can check the version of the browser and switch various features of your scripts on and off accordingly.

With a little thought and planning, you can design your script so that it degrades graceful if it is run on less capable browser versions than that for which you originally designed it.

The differences between browsers are now so complex and so diverse that it is difficult to encompass them all in a single reference source. This book is structured to allow it to be revised on a component-by-component basis so that where browsers differ from one another, the granularity of the book is approximately the same and can track those differences as they become known.

The differences between the browsers may change in very subtle ways even with minor browser version changes. We concentrate on the differences between major versions and use annotations to cover important differences between minor browser versions.

Any feedback or observations you care to submit will be welcomed, tested, and added to the future editions of the book.

Good workaround techniques involve innovative use of scripts to create your own properties and methods to emulate missing functionality. For example, the `window.opener` property is not available on all versions of Netscape. You could create a property of your own that refers to the parent window when a new window object is created. If that property is always present and created under script control, then you can use that property rather than the one that may or may not be present in the built-in object model. This is generally more robust, but may not exploit the very latest features of the available browsers.

See also:	Compatibility, `Date` object, Defensive coding, Internet Explorer, `Window.opener`

Browser wars (Definition)

The contest between browser manufacturers to gain dominance in the market.

As this is being written, it is clear that Microsoft has won the war of the browsers – for now at least. The Netscape browser has lost market share, to the extent that it is fast becoming a minority browser.

This poses an interesting situation, in that Microsoft has sufficient market share that it can perhaps reduce the effort that it puts into browser support.

Actually, it is at such a time that it should put even more resources into it. That is because, now that it is so dominant, it should be obliged to make sure its browser is supported identically on every platform it is available on, and make it available on any remaining platforms.

Whether it will do this is open to question as it could detract from its dominance of the operating system marketplace.

This conflict of interests is potentially damaging for the end-user and the web developer.

Right at this moment, there is a significant proportion of the feature set in MSIE that is not supported on platforms other than Windows.

Granted, it is acceptable that COM and ActiveX cannot easily be provided on non-Windows platforms, but the CSS support should be identical, as should the integration with clipboards and other parts of the OS where it is possible.

Netscape 6.0 has just been released in its final form as this is being written. The new version is so radically different as to classify it as being a different browser. Its internal document model follows the DOM specification very closely. Netscape had adhered to the DOM specified class names where Microsoft has not, even though it has constructed a DOM representative object model in the browser.

Maybe Netscape can win back some proportion of the users it has lost to Microsoft in the last few years. However, there is still much to be done to correct some shortcomings in the released quality of the new Netscape browser.

btoa() (Method)

Used to encode some data into base-64 form.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	`btoa(aBinaryString)`
	N	`myWindow.btoa(aBinaryString)`
Argument list:	`aBinaryString`	A string of binary data to be encoded

See also:	`Window.atob(),Window.btoa()`

Built-in function (Definition)

Functions that are part of the core JavaScript implementation.

Availability:	ECMAScript edition – 2

Built-in functions are implemented as `Function` objects.

Examples of built-in functions are `parseInt()` and `Math.exp()`. These are functions provided by the `Global` object and the `Math` object respectively. They may be referred to as built-in methods in some documentation.

None of the built-in functions implement the internal `Construct()` method and therefore they cannot be used with the new operator to create another instantiation.

Generally, none of the built-in functions will have a `prototype` property, but since they cannot be instantiated this should not cause any problems.

Built-in function objects have a `length` property whose value is an integer. This generally indicates the number of arguments the function expects to be supplied with. Sometimes functions may be supplied with optional arguments. The length value returns the maximum number of arguments that are expected. The `length` property of a built-in function has the `ReadOnly`, `DontDelete` and `DontEnum` attributes set for it.

Generally, all the other properties of a built-in function have the `DontEnum` attribute set.

See also:	`Function` object,Native object

Cross-references:

ECMA 262 edition 2 – section – 15

ECMA 262 edition 3 – section – 15

Built-in method (Definition)

Object methods that are provided as part of the base JavaScript implementation.

Refer to:

Built-in function

Built-in object (Definition)

Objects that are part of the core JavaScript implementation.

Availability:	ECMAScript edition – 2

A built-in object is provided by the core interpreter independently of the host environment.

Built-in objects are available at the outset of script execution and do not need to be created. They are all native objects. Additional built-in objects may be added by the implementation over and above those specified by the core functionality in the language specification.

See also:	Native object

Cross-references:

ECMA 262 edition 2 – section – 4.3.7

ECMA 262 edition 3 – section – 4.3.7

Button object (Object/DOM)

An object representing an `<INPUT TYPE="button">` HTML button in a form.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Inherits from:	`Input` object	
JavaScript syntax:	-	`myButton = myDocument.aFormName.anElementName`
	-	`myButton = myDocument.aFormName.elements [anItemIndex]`
	IE	`myButton = myDocument.all.anElementID`
	IE	`myButton = myDocument.all.tags("INPUT") [anIndex]`
	IE	`myButton = myDocument.all[aName]`
	-	`myButton = myDocument.forms[aFormIndex]. anElementName`
	-	`myButton = myDocument.forms[aFormIndex]. elements[anItemIndex]`
	-	`myButton = myDocument.getElementById (anElementID)`
	-	`myButton = myDocument.getElementsByName (aName)[anIndex]`
	-	`myButton = myDocument.getElementsByTagName ("INPUT")[anIndex]`
HTML syntax:	`<INPUT TYPE="button">`	
Argument list:	`anItemIndex`	A valid reference to an item in the collection
	`aName`	The `name` attribute of an element
	`anElementID`	The `ID` attribute of an element
	`aFormIndex`	A reference to a single form in the `forms` collection
	`anIndex`	A valid reference to an item in the collection
Object properties:	`type`, `value`	
Object methods:	`handleEvent()`	
Event handlers:	`onAfterUpdate`, `onBeforeUpdate`,`onBlur`, `onClick`, `onDblClick`, `onErrorUpdate`, `onFilterChange`, `onFocus`,`onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onRowEnter`, `onRowExit`	

Many properties, methods and event handlers for this object are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

There isn't really a `Button` object class in Netscape, but it is helpful when trying to understand the wide variety of input element types if we can reduce the complexity by discussing only the properties and methods of a button. In actual fact, the object is represented as an item of the `Input` object class.

In MSIE, there is a special `BUTTON` class that is used to represent a `<BUTTON>` tag. It is documented separately in its own topics. The `Button` object is the correct spelling for a DOM level 1 compliant implementation.

Event handling support via properties containing function objects was added to `Button` objects at version 1.1 of JavaScript.

Warnings:

- Note that on MSIE, `Input` objects are actually `INPUT` objects, because MSIE follows a general rule of naming object classes after the capitalized name of the HTML tag that instantiates them. However, in some special cases, MSIE creates other object types. For buttons, it uses the `BUTTON` class.
- Netscape does not support the `defaultValue` property for this sub-class of the `Input` object.

See also:	`Element` object, `Element.isTextEdit`, `Form.elements[]`, `FormElement` object, `Input` object, `Input.accessKey`, `onClick`, `TextRange` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`type`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onClick`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onErrorUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onRowEnter`	-	3.0 +	-	4.0 +	-	-	-	-
`onRowExit`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

Button.handleEvent() (Method)

Passes an event to the appropriate handler for this object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`myResetButton.handleEvent (anEvent)`
Argument list:	`anEvent`	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`handleEvent()`

Button.type (Property)

The type value for the object.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myButton.type`
HTML syntax:	`<INPUT TYPE="button">`	

The value of this property will be "`button`" when the `<INPUT>` HTML tag describes a form `Button` object.

See also:	`Input.type`

Property attributes:

`ReadOnly`.

Button.value (Property)

The text string displayed in the button.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myButton.value`

The value of a button is also used to place a legend into the button image on screen.

Warnings:

- This property may be changed on some platforms, but not others.

See also:	`Input.value`

BUTTON object (Object/HTML)

An object that represents a special MSIE `<BUTTON>` element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`myBUTTON = myDocument.all.anElementID`
	IE	`myBUTTON = myDocument.all.tags("BUTTON") [anIndex]`
	IE	`myBUTTON = myDocument.all[aName]`
	-	`myBUTTON = myDocument.getElementById (anElementID)`
	-	`myBUTTON = myDocument.getElementsByName (aName)[anIndex]`
	-	`myBUTTON = myDocument.getElementsByTagName ("BUTTON")[anIndex]`
HTML syntax:	`<BUTTON> ... </BUTTON>`	
Argument list:	`anIndex`	A valid reference to an item in the collection
	`aName`	The `NAME` attribute of an element
	`anElementID`	The `ID` attribute of an element
Object properties:	`accept`, `accessKey`, `alt`, `dataFld`, `dataFormatAs`, `dataSrc`, `form`,`name`, `status`, `tabIndex`, `type`, `value`	
Object methods:	`createTextRange()`	
Event handlers:	`onAfterUpdate`, `onBeforeUpdate`, `onBlur`, `onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onFocus`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onResize`,`onRowEnter`, `onRowExit`,`onSelectStart`	

This is an additional kind of button object, over and above that provided for the `<INPUT TYPE="Button">` tag.

Warnings:

- This object is not the same as a `Button` object, which is a convenience class that is really instantiated as an `Input` object. Netscape only supports `Button` (Input) objects and does not support `BUTTON` objects. MSIE supports both. The properties of each type of button object are different.

See also:	`Element` object, `Element.isTextEdit`, `Form.elements[]`, `FormElement` object,`Input` object, `Input.accessKey`, `onClick`,`TextRange` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accept	-	5.0 +	-	5.0 +	-	-	-	Warning
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
alt	-	3.0 +	-	4.0 +	-	-	-	-
dataFld	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
dataFormatAs	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
dataSrc	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
form	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
name	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
status	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
type	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
value	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
createTextRange()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onResize	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

BUTTON.accept (Property)

Defines an acceptable MIME type to be submitted to the server. Not supposed to be supported by the BUTTON class.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	myBUTTON.accept

Refer to the MIME types topic for a list of MIME types to use in this property.

Warnings:

- This property is inherited, but is not applicable to a BUTTON object.

See also:	MIME types

BUTTON.alt (Property)

The tool-tip text for the BUTTON object.

Availability:	JScript – 1.0 Internet Explorer – 3.02	
Property/method value type:	String primitive	
JavaScript syntax:	IE	myBUTTON.alt

Objects can have an alternative text string associated with them. This is especially useful on browsers that cannot cope with the tag, in which case they may display the alternative text. If spoken styles are supported, the text may be read out to the user. Some browsers will also display the text as a tool-tip if the mouse is positioned over the object and remains static for a few seconds.

BUTTON.name (Property)

This corresponds to the NAME attribute of the <BUTTON> HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	myBUTTON.name

Objects are identified either by the `NAME` tag attribute or by the `ID` tag attribute. MSIE seems slightly better disposed towards the `ID` attribute than earlier versions of Netscape. However, in many cases, both browsers support either technique and in some cases will locate items named with either attribute as if they existed in a single namespace. Version 6.0 of Netscape may restore parity with MSIE in this respect.

See also:	`Input.name`

BUTTON.type (Property)

The type of this `BUTTON` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBUTTON.type`

The property should contain one of the following values:

- `button`
- `reset`
- `submit`

See also:	`Input.type`

Property attributes:

`ReadOnly`.

BUTTON.value (Property)

The value of this `BUTTON` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myBUTTON.value`

The value of a button is also used to place a legend into the button image on screen.

See also:	`Input.value`

By reference (Definition)

A means of passing data to functions, methods, and property accessors.

Passing a value by reference means that you pass a pointer to the data value. This means that when you copy the reference to another variable, you then have two variables pointing at the same data.

The data for both is identical. Modifying the data value pointed at by one, changes the value for the other variable also.

Non-primitive values (Objects) are passed in this manner.

Note that although JavaScript has a good pass-by-reference technique built-in to the implementation, it does not support pointers as a specific data type as the C language does.

Passing a value by reference into a function allows the function to make changes that are visible outside the function.

In JavaScript, you cannot manipulate these parameter passing mechanisms. It will choose to pass by value or by reference according to the data type of the value being put into the argument. This is often the result of evaluating an expression. JavaScript takes care of all the type conversions for you.

By value (Definition)

A means of passing data to functions, methods, and property accessors.

Passing data by value means that the data itself is stored in the variable. Assigning one variable to another copies the value.

Changing the value in one variable leaves the other unaffected.

Non-object values (primitives) are generally passed in this manner.

Passing data by value to a function means that the function cannot affect the value outside the function since it only has a copy of the value to work on.

In JavaScript, you cannot manipulate these parameter passing mechanisms. It will choose to pass by value or by reference according to the data type of the value being put into the argument. This is often the result of evaluating an expression. JavaScript takes care of all the type conversions for you.

byte (Reserved word)

Reserved for future language enhancements.

A byte is a set of 8 adjacent binary digits (bits). It is big enough to hold an 8 bit character code, which will support the subset of 16 bit Unicode characters that most JavaScript users are likely to need, at least for developing scripts for use with the English language.

The least significant bit is called the low-order bit and the most significant bit is called the high-order bit. These do not necessarily map one to one to the bits stored in the memory of the computer, which may be big-endian or little-endian. This is thankfully hidden from the JavaScript programmer who will need to operate on a standardized IEEE-754 bit pattern when working with binary values stored in Numeric primitive values.

The fact that the ECMAScript standard (edition 2) reserves this word for future use, suggests that some byte level support is expected to be added to the language at some time in the future.

This keyword also represents a Java data type and the `byte` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	`char`,IEEE 754, LiveConnect, Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Calendar time (Definition)

Time values measured according to the Gregorian calendar.

Time values are one of the areas of least portability between languages and systems. Although many languages make a brave attempt to provide portable facilities, it is rarely indeed that you will find two systems that can sensibly exchange the binary or internal time values.

You can transfer time values between systems but you'll need to do it like this. First, you will need to convert from internal time values to some textual representation that can then be scanned and parsed back into an internal value on the other system. Then transmit the string to the other end and invoke some local processing there to convert the time value back in its internal form.

There are a variety of names for this internal time value, one of which is Calendar time.

See also:	Broken down time, Date and time, `Date` object, Daylight savings time adjustment, Local time, Time range, Time value, Universal coordinated time

Call (Function/internal)

An internal mechanism for executing function calls.

Availability:	ECMAScript edition –2

This is the internal mechanism by which functions are implemented.

Objects supporting this method are called functions.

When they are called, they add themselves to the scope chain and any variables subsequently declared are added to that scope. Hence local objects belong to the function being executed.

Another name for the function being executed is the `call` object.

Warnings:

- The `Global` object does not have a `Call` property and therefore you cannot use it as a function.

See also:	`Function` property, Internal Method, `JSObject.call()`

Property attributes:

`DontEnum`, `Internal`.

Cross-references:

ECMA 262 edition 2 – section – 8.6.2

ECMA 262 edition 3 – section – 8.6.2

Call a function (Definition)

To invoke a function during script execution.

See also:	Call by reference, Call by value, `JSObject.call()`, Function call

Call by reference (Definition)

Calling functions and passing references to receiving LValues in the arguments.

If you want to modify a value that is passed to a function, you need to pass a reference to it. Normally you would depend on a function returning a single value.

You can do this by creating an object, passing the object but mutating the values of the object's properties. This bridges the scope chain because the locally scoped copy refers to the outer scoped LValue.

This is demonstrated in the example.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myObject = new Object();
myObject.myVar = 22;
callMe(myObject);
```

```
document.write("<BR>");
document.write(myObject.myVar);
function callMe(aValue)
{
   document.write(aValue.myVar);
   document.write("<BR>");
   aValue.myVar = 100;
   document.write(aValue.myVar);
}
</SCRIPT>
</BODY>
</HTML>
```

Call by value (Definition)

Calling functions and passing values in the arguments.

When you call by value, you are passing an immutable constant to a function. The function will create a copy locally and that copy will be accessible in a locally scoped variable having the name of the formal parameter.

However, on exit the local value is discarded leaving the original unchanged. It doesn't matter whether you pass a literal constant value or a variable containing a value. This is easier to understand if you already have a good grasp of the scope chain mechanism.

The example demonstrates this.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myVar = 22;
callMe(myVar);
document.write("<BR>");
document.write(myVar);
function callMe(aValue)
{
   document.write(aValue);
   document.write("<BR>");
   aValue = 100;
   document.write(aValue);
}
</SCRIPT>
</BODY>
</HTML>
```

See also: Scope chain

Call object (Object/internal)

The currently executing function is a `call` object.

Availability:	ECMAScript edition – 2
See also:	`Arguments` object, Function scope, `Function.arguments[]`, `JSObject.call()`,`Call`

Cross-references:

ECMA 262 edition 2 – section – 8.6.2

ECMA 262 edition 3 – section – 8.6.2

Call-back event (Definition)

A mechanism for creating frameworks that call user-supplied functions.

See also:	Event, `JSObject.call()`, Plugin events, Event handler

Calling event handlers (Definition)

Event handlers can be called in many different ways.

If you implement an event handler as a function, then you can call it from other functions as needed. For example, we can build a form validator and associate it with the `onSubmit` event for the form.

We might want to invoke that validator when something else changes on the page or as a result of the user clicking on various buttons. Because the submission is still handled by the browser, invoking the form validation event handler won't submit the form unless it is called as a result an `onSubmit` event being triggered.

The form only gets submitted to the server if the form validator returns the correct Boolean flag value when it is invoked in response to an `onSubmit` trigger event.

There is another way to submit the form's contents. That is by calling the `submit()` method belonging to a `Form` object. That doesn't trigger an `onSubmit` event if it is called within the context of an `onSubmit` event handler. You could use this technique if you wanted the form contents to be submitted by some action other than clicking on a submit button.

See also:	Event handler, Event handler properties, Function call

CanPut() (Function/internal)

Internal private function.

Availability:	ECMAScript edition – 2

This internal function returns a Boolean value to indicate whether the named property can be changed in the containing object.

If the property is found, the value of its `ReadOnly` attribute is checked. If it has a `ReadOnly` attribute, the result of `CanPut()` must be `false`. Otherwise, having found the property, the `true` result will be returned.

If the property does not exist in the receiving object, the prototype chain is walked until the property or a null prototype is encountered. At each inheritance level, the `CanPut()` function is used to determine the existence of the property.

If a null prototype is encountered, the result will be `true`, since the property can then be created in the original receiving object.

If the prototype is a `host` object that does not implement the `CanPut()` function, then `false` is returned as a result.

Because the prototype chain is walked extensively by the `CanPut()` function, if the prototype chain is not finite and terminated with a null at some stage, a recursive loop is built and the function never returns.

See also:	Internal Method

Property attributes:

`Internal`.

Cross-references:

ECMA 262 edition 2 – section – 8.6.2.3

ECMA 262 edition 3 – section – 8.6.2.3

CAPTION object (Object/HTML)

An object that represents the `<CAPTION>` HTML tag, which is used inside a `<TABLE>`.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Inherits from:	`Element` object

JavaScript syntax:	IE	*myCAPTION* = *myDocument*.all.*anElementID*
	IE	*myCAPTION* = *myDocument*.all.tags("CAPTION")[*anIndex*]
	IE	*myCAPTION* = *myDocument*.all[*aName*]
	-	*myCAPTION* = *myDocument*.getElementById(*anElementID*)
	-	*myCAPTION* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myCAPTION* = *myDocument*.getElementsByTagName("CAPTION") [*anIndex*]
HTML syntax:	<CAPTION> ... </CAPTION>	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The *name* attribute of an element
	anElementID	The *ID* attribute of an element
Object properties:	align, vAlign, align, vAlign	
Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onChange, onClick, onDblClick, onDragStart, onErrorUpdate, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onScroll, onSelect, onSelectStart	

The caption forms an integral part of the table to which it belongs. It needs to be defined inside the <TABLE> tags.

The DOM level 1 standard describes these objects as TableCaptionElement objects.

See also:	Element object, style.captionSide, TABLE object, TABLE.caption, TABLE.createCaption(), TABLE.deleteCaption()

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, Deprecated
vAlign	-	3.0 +	-	4.0 +	-	-	-	Warning, Deprecated
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, Deprecated
vAlign	-	3.0 +	-	4.0 +	-	-	-	Warning, Deprecated

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onChange	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	-
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onSelect	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

CAPTION.align (Property)

The horizontal alignment of the caption with respect to its parent table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myCAPTION*.align

The alignment of the `CAPTION` object with respect to its containing parent table object is defined in this property. The expected and widely available set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`

- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

These may not all be supported by all browsers.

Note also that this property is deprecated in favor of using style properties such as `textAlign` and `verticalAlign`, which correspond to the style sheet attributes `text-align` and `vertical-align`.

Warnings:

- Browsers may not always render the caption identically. The left and right values in particular are rendered differently between MSIE and Netscape. MSIE draws the caption at the top and bottom but takes the left and right as a justification control. Netscape places the caption to the left or right of the table.
- Currently MSIE transgresses the rules laid down by the HTML 4.0 standard.

See also:	`style.captionSide`, `style.textAlign`, `style.verticalAlign`

CAPTION.vAlign (Property)

The vertical alignment of a caption with respect to its parent table.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myCAPTION*`.vAlign`

This is only supported by MSIE and is intended to provide for alignment correction, which the `align` property handles incorrectly with respect to the HTML 4.0 standard.

This property is basically supported for backwards compatibility.

The `vAlign` property may be set to these values:

- `bottom`
- `top`

Warnings:

- It is probably wise to use the `<CAPTION>` tag with discretion and care. You should probably do some cross-browser testing to ensure its placement is what you intended.

See also:	`style.captionSide`

captureEvents() (Function)

Part of the Netscape 4 event propagation complex.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`captureEvents(`*`anEventMask`*`)`
	N	*`myObject`*`.captureEvents(`*`anEventMask`*`)`
	N	*`myWindow`*`.captureEvents(`*`anEventMask`*`)`
Argument list:	*`anEventMask`*	A mask constructed with the manifest event constants

Warnings:

- Since a bit mask is being used, this must be an int32 value. This suggests that there can only be 32 different event types supported by this event propagation model.
- This capability is deprecated and is not supported in Netscape 6.0 any more. It never was supported by MSIE, which implements a completely different event model. As it turns out, the DOM level 2 event model converges on the MSIE technique.

See also:	`Document.captureEvents()`, `Document.releaseEvents()`, `Element.onevent`, Event handler, Event management, Event propagation, Event type constants, `Frame` object, `handleEvent()`, Keyboard events, `Layer.captureEvents()`, `Layer.releaseEvents()`, `onLoseCapture`, `onMouseMove`, `Window` object, `Window.captureEvents()`, `Window.releaseEvents()`, `Window.routeEvent()`

Cross-references:

Wrox *Instant JavaScript* – page – 55

case ... : (Label)

Part of the `switch ... case` mechanism. The `case` keyword denotes a label associated with one of the selectors.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server – 3.0

See also:	`break`, Flow control, Label, Selection statement, `switch( ... ) ... case: ... default: ...`

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

ECMA 262 edition 3 – section – 12.11

Case Sensitivity (Definition)

Upper and lower case are not identical when used in identifiers.

Availability:	ECMAScript edition – 2

Identifiers in JavaScript are case-sensitive.

This means that, for example, variables will refer to distinctly different values if they differ in the case of any part of their names. Hence `aaa` is not the same variable as `Aaa`.

MSIE browsers prior to version 4 were less particular about case-sensitivity. Since the ECMA standard requires strict case-sensitive behavior, this is now the norm.

Some early versions of the WebTV set top box prior to the Summer 2000 release also lacked case-sensitive behavior regarding built-in method and property names.

Warnings:

- In MSIE version 3, all client-side object and property names were case-insensitive. Beware of any old scripts, which may have worked on MSIE version 3, but don't work on later browsers.

- JavaScript style sheets in Netscape 4 are also case-insensitive.

See also:	JavaScript Style Sheets, JellyScript

Cross-references:

ECMA 262 edition 2 – section – 7.5

ECMA 262 edition 3 – section – 7.6

O'Reilly *JavaScript Definitive Guide* – page – 27

Cast operator (Definition)

A way of converting data types.

Primitive values can be converted from one type to another or rendered as objects by using object constructors to convert the values.

Type Name:	Description:
Aggregate	A collection of atomic types assembled collectively into an object.
Arithmetic	All types that yield a value that can be operated on numerically.
Array	Collections of objects and identifiers assembled into a sequence.
Basic	The fundamental simple, non-object types.
Boolean	This type can store and yield true or false values.
Completion	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
List	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
Null	This has exactly one value, null, and is distinct from undefined.
Number	Integer and floating-point values are all stored in a generic number type.
Object	An object is an unordered collection of properties. Each property consists of a name, a value, and a set of attributes.
Reference	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
Scalar	The non-object types.
String	Strings are arrays of characters that are accessible individually by indexing their position in the sequence.
Undefined	This value is returned by variables that have not yet been assigned a value.

In compiled languages, this is called casting. In JavaScript the values are actually converted using methods and function calls on the objects. This yields a new value rather than making the old one look like the new type that is required. It is really conversion rather than casting.

See also:	Aggregate type, Array simulation, Escape sequence (\), `escape()`, `MakeDate()`, `MakeDay()`, `MakeTime()`, Native object, null, Operator Precedence, Primitive value, `unescape()`

catch(...) (Function)

Part of the `try ... catch ...finally` error-handling mechanism.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	`catch(`*`anError`*`)`
Argument list:	*`anError`*	An instance of the error object

The ECMAScript standard (edition 2) defined the `catch` keyword and reserves it for future use. Edition 3 mandates that this should now be supported in a compliant interpreter.

In anticipation of that, it is available in JavaScript version 1.4. This is also now supported in JScript version 5.0 as well.

Refer to the `try ... catch ... finally` topic for more details.

See also:	`Error` object, `EvalError` object, Exception handling, `finally ...`, `RangeError` object, `ReferenceError` object, `SyntaxError` object, `throw`, `try ... catch ... finally`, `TypeError` object, `URIError` object

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

ECMA 262 edition 3 – section – 12.14

Category of an object (Definition)

Why we categorize topics the way we do.

Objects, properties and methods fall into many categories. It helps to understand why an item is available and what it does if you can map it into a particular context. Here is a summary of the different categories we use:

Category:	Description:
Browser	Language elements that are really only useful on the browser side.
Core	Language elements that form part of the core of the language.
Desktop	Language facilities provided to aid desktop automation.

Table continued on following page

Category:	Description:
Embedded	Language facilities particularly intended for use in embedded interpreters.
Generic	Topics that discuss something that could be used server-side or client-side are categorized as generic. Properties and objects may be defined server-side and deployed client-side.
Microsoft	There are many proprietary additions by Microsoft. These are the particularly noteworthy items and are generally non-portable.
Server	Language elements that are really only useful on the server side.
Shell	Language facilities that are added for use in shell scripts.

CDATASection object (Object/DOM)

Part of the extended interface that DOM describes for supporting non-HTML content.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Inherits from:	`textNode` object
JavaScript syntax:	- *myCDATASection* = *myDocument*.createCDATASection(*someData*)
Argument list:	*someData* — The data content for the new object

The extended interface supports various document forms other than HTML. This object is used to encapsulate marked up XML without needing to escape all of the markup characters.

You can test for the availability of this feature by means of the `Implementation.hasFeature()` method. In this case, test for a feature name of "`XML`" and a version value of "`1.0`"

See also:	`Document.createCDATASection()`

Inheritance chain:

`CharacterData` object, `Node` object, `textNode` object

CENTER object (Object/HTML)

An object that represents the `<CENTER>` HTML tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Inherits from:	`Element` object

JavaScript syntax:	IE	*myCENTER* = *myDocument*.all.*anElementID*
	IE	*myCENTER* = *myDocument*.all.tags("CENTER")[*anIndex*]
	IE	*myCENTER* = *myDocument*.all[*aName*]
	-	*myCENTER* = *myDocument*.getElementById(*anElementID*)
	-	*myCENTER* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myCENTER* = *myDocument*.getElementsByTagName ("CENTER")[*anIndex*]
HTML syntax:	<CENTER> ... </CENTER>	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The *name* attribute of an element
	anElementID	The *ID* attribute of an element
Object methods:	removeAttribute()	
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
removeAttribute(0	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

Refer to:

`Element` object

.cfg (File extension)

A configuration file for Netscape.

See also:	`netscape.lck`, Preferences

.cgi (File extension)

Common gateway interface dynamic page.

Refer to:

File extensions

CGI-Driven JavaScript (Definition)

Using JavaScript in the request - response loop of a web server.

You can use JavaScript to generate the response to a web browser's incoming requests at the web server.

This is a useful and powerful way to extend the CGI capabilities of your web server. Forms handling with products such as ScriptEase WSE (Web Server Edition) are streamlined due to the interpreter having additional features that usefully package the request data before your script is called.

See also:	Host environment, Platform, Server-side JavaScript, Shell Scripting with JavaScript

Cross-references:

Wrox *Instant JavaScript* – page – 5

char (Reserved word)

Reserved for future language enhancements.

The ECMAScript (edition 2) reserves the `char` keyword for future use. This suggests some additional C-like functionality may be added in the future. A `char` may be represented by a byte.

However in JavaScript, characters are really double-byte values since they encode a Unicode code point in each character.

This keyword also represents a Java data type and the `char` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	byte, `java.lang.Character`, LiveConnect, Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

Character constant (Definition)

A literal description of a single character.

Property/method value type:	`String primitive`

A character constant in JavaScript is represented by a single character length String primitive. Technically it is a string, not a character. In JavaScript the single quote delimiter encloses strings as well as the double quote delimiter.

Beware of this if you are familiar with C language character constants. In C, the single quote (apostrophe) is used to enclose a single character that can be represented by a byte.

Some character constant escape codes are listed in the following table:

Escape Sequence:	Name:	Symbol:
`\"`	Double Quote	"
`\'`	Single Quote (Apostrophe)	'
`\\`	Backslash	\
`\a`	Audible alert (MSIE displays the letter a)	<BEL>
`\b`	Backspace (ignored silently in MSIE)	<BS>
`\f`	Form Feed (ignored silently in MSIE)	<FF>
`\n`	Line Feed (Newline - MSIE inserts a space)	<LF>
`\r`	Carriage Return (MSIE inserts a space)	<CR>
`\t`	Horizontal Tab (MSIE inserts a space)	<HT>
`\v`	Vertical tab (MSIE displays the letter v)	<VT>
`\0nn`	Octal escape	-
`\042`	Double Quote	"
`\047`	Single Quote (Apostrophe)	'
`\134`	Backslash	\
`\xnn`	Hexadecimal escape	-

Table continued on following page

Escape Sequence:	Name:	Symbol:
\x22	Double Quote	"
\x27	Single Quote (Apostrophe)	'
\x5C	Backslash	\
\unnnn	Unicode escape	-
\u0022	Double Quote	"
\u0027	Single Quote (Apostrophe)	'
\u005C	Backslash	\
\uFFFE	A special Unicode sentinel character for flagging byte reversed text	-
\uFFFF	A special Unicode sentinel character	-

See also:	Character handling, Constant, Constant expression, Escape sequence (\), Literal, Primitive value

Character display semantics (Definition)

How characters are displayed on the implementation's console.

Although the standard defines many escape sequences (see table), how these are displayed depends very much on the way that the implementation uses the output of JavaScript:

Escape Sequence:	Name:	Symbol:
\"	Double Quote	"
\'	Single Quote (Apostrophe)	'
\\	Backslash	\
\a	Audible alert (MSIE displays the letter a)	<BEL>
\b	Backspace (ignored silently in MSIE)	<BS>
\f	Form Feed (ignored silently in MSIE)	<FF>
\n	Line Feed (Newline - MSIE inserts a space)	<LF>
\r	Carriage Return (MSIE inserts a space)	<CR>
\t	Horizontal Tab (MSIE inserts a space)	<HT>
\v	Vertical tab (MSIE displays the letter v)	<VT>
\0nn	Octal escape	-
\042	Double Quote	"
\047	Single Quote (Apostrophe)	'
\134	Backslash	\
\xnn	Hexadecimal escape	-
\x22	Double Quote	"
\x27	Single Quote (Apostrophe)	'

Table continued on following page

Escape Sequence:	Name:	Symbol:
\x5C	Backslash	\
\unnnn	Unicode escape	-
\u0022	Double Quote	"
\u0027	Single Quote (Apostrophe)	'
\u005C	Backslash	\
\uFFFE	A special Unicode sentinel character for flagging byte-reversed text	-
\uFFFF	A special Unicode sentinel character	-

Encoding line feeds, form feeds, and tabs into data that ultimately gets output as part of a `document.write()` method suggests that the target is an HTML page. When HTML is rendered, any embedded tabs and line terminators have no effect at all on the displayed output apart from some undesirable side effects in older browsers, which used to display line terminators inside anchor tags in a very odd way.

On the other hand, JavaScript that is generating a text data stream that is going to be returned via a TCP socket may well want to encode all kinds of escaped control characters.

Warnings:

- Generally a browser will ignore any escape sequences it cannot cope with. Some it will ignore silently such as a `\b` which results in no output in the MSIE browser. For others, such as `\a`, MSIE ignores the backslash but writes the letter 'a' into the document output. A few escape characters result in a space character being inserted into the output text.
- You should, as a matter of course, clean your HTML text of any unwanted escape characters if you can.

See also:	Character set, Environment, Escape sequence (\)

Character entity (Definition)

A means of escaping difficult-to-type characters for use in HTML.

Refer to:

HTML Character entity

Character handling (Advice)

Functions for testing character attributes.

Developers who use the C language and who are converting to JavaScript may be used to having support for testing various properties of character codes.

These functions are not formally part of the JavaScript language, although some of them are provided as part of the host environment through additional objects that provide C-like functionality. These are modeled on the `Math` object and cannot usually be instantiated by belonging to the `Global` object in the same way as the `Math` object does.

ScriptEase by Nombas is one interpreter that provides support for C language functionality through its `Clib` object.

If you are using other interpreters, you can simulate these character handling functions with fragments of script and some bitwise operators.

The following functions that are normally available to C programmers are simulated with the script examples in the following sections:

- `isalnum()`
- `isalpha()`
- `iscntrl()`
- `isdigit()`
- `isgraph()`
- `islower()`
- `isprint()`
- `ispunct()`
- `isspace()`
- `isupper()`
- `isxdigit()`

See also:	`isAlnum()`,`isAlpha()`,`isCtrl()`,`isDigit()`,`isGraph()`, `isLower()`,`isODigit()`,`isPrint()`,`isPunct()`,`isSpace()`, `isUpper()`,`isXDigit()`

Character set (Definition)

The collection of characters that the script can operate on.

Since JavaScript 1.3 and JScript 3.0, the language has been built around the Unicode standard. This means its identifiers and hence its script source code is intended to be represented by a sequence of Unicode characters. The benefit of this is that identifiers can be named using international characters. The reality is that some implementations don't support this very well, even if they can parse and process Unicode correctly as data.

As is the case with many languages, there may be a character set that can be used for data and a smaller sub-set that is valid for use when editing script source text.

Strictly speaking, a JavaScript script source can be encoded with 7 bit ASCII characters since there are mechanisms to escape generated character codes that are multi-byte Unicode code points.

The Unicode standard describes a large number of international character sets in terms of the character glyphs supported by them. There are also a large number of ISO standardized character sets.

See also:	ASCII, Character display semantics, Character handling, Character-case mapping, Environment, Escape sequence (\), `isLower()`,`isUpper()`, Locale-specific behavior, Localization, Multi-byte character, Unicode

Character testing (Definition)

Testing characters for attributes.

The following functions that are normally available to C programmers are simulated with script examples in the following sections:

- `isalnum()`
- `isalpha()`
- `iscntrl()`
- `isdigit()`
- `isgraph()`
- `islower()`
- `isprint()`
- `ispunct()`
- `isspace()`
- `isupper()`
- `isxdigit()`

Strictly speaking, these functions should be coded to be aware of locale-specific issues. You may want to take example simulations provided here and modify them to your own needs to support that. These are just basic working examples.

See also:	Character handling, Character-case mapping, `isAlnum()`, `isAlpha()`, `isCtrl()`, `isDigit()`, `isGraph()`, `isLower()`, `isODigit()`, `isPrint()`, `isPunct()`, `isSpace()`, `isUpper()`, `isXDigit()`, `String.charAt()`, `String.charCodeAt()`, `String.fromCharCode()`

Character value (Definition)

A numeric value based on the Unicode and ASCII character code points.

See also:	ASCII, Integer constant, Unicode

Character-case mapping (Overview)

Character case conversion.

The conversion of characters from upper to lower case and vice versa is accomplished in JavaScript by means of the `String` object. This provides two methods that can be applied to a `String` object to change its case. However, this would not work on String primitives so you may need to do an object conversion first.

The ECMAScript standard mandates that only the base characters need be mapped between the upper and lower case. Sorting and case conversion may support other international characters in some implementations, but this is not covered by the standard.

Localization issues may affect this sort of operation.

The interpreter should automatically convert any String primitives to `String` objects so that the method can be applied. This means that this should work:

```
"aaaa".toUpperCase()
```

And you should not need to do this:

```
String("aaaa").toUpperCase()
```

See also:	ASCII, Character handling, Character set, Character testing, `isLower()`, `isUpper()`, Locale-specific behavior, `String.charCodeAt()`, `String.fromCharCode()`, `String.toLocaleLowerCase()`, `String.toLocaleUpperCase()`, `String.toLowerCase()`, `String.toUpperCase()`, Unicode

CharacterData object (Object/DOM)

A sub-class of the `node` object with extensions to support access to character data within the object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Inherits from:	`Node` object	
JavaScript syntax:	-	*myCharacterData* = new CharacterData()
Object properties:	`data`, `length`	
Object methods:	`appendData()`, `deleteData()`, `insertData()`, `replaceData()`, `substringData()`	

See also:	`COMMENT` object

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`data`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`length`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`appendData()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`deleteData()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`insertData()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`replaceData()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`substringData()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Inheritance chain:

`Node` object

CharacterData.appendData() (Method)

Append some text to the end of the character data.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myCharacterData*.appendData(*aString*)
Argument list:	*aString*	Some data to append

CharacterData.data (Property)

The current contents of the character data node.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myCharacterData*.data

CharacterData.deleteData() (Method)

Remove a section of text from the data contained in the character data node.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myCharacterData*.deleteData(*anOffset*, *aCount*)
Argument list:	*anOffset*	The start of the deleted section
	aCount	The length of the deleted section

CharacterData.insertData() (Method)

Insert some additional text into the character data node.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myCharacterData*.insertData(*anOffset*, *aString*)
Argument list:	*anOffset*	A location to insert the data at
	aString	The data to insert

CharacterData.length (Property)

Return the length (in characters) of the character data node.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	Number primitive
JavaScript syntax:	- *myCharacterData*.length

CharacterData.replaceData() (Method)

Replace a section of text in the character data node with some new text.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myCharacterData*.replaceData(*anOffset*, *aCount*, *aString*)
Argument list:	*anOffset*	The location where the replacement starts
	aCount	The length of data to be replaced
	aString	The new data to insert

CharacterData.substringData() (Method)

Non destructively extract a section of the text from the character data node.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myCharacterData*.substringData(*anOffset*, *aCount*)
Argument list:	*anOffset*	A location where the substring starts
	aCount	The length of the substring

Checkbox object (Object/DOM)

A checkbox to be used in a form. It toggles as it is clicked, but is not related to other checkboxes in the way that radio buttons are related to one another in families.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Inherits from:	`Input` object	
JavaScript syntax:	-	*myCheckbox* = *myDocument*.*aFormName*.*anElementName*
	-	*myCheckbox* = *myDocument*.*aFormName*.elements[*anItemIndex*]
	IE	*myCheckbox* = *myDocument*.all.*anElementID*
	IE	*myCheckbox* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*myCheckbox* = *myDocument*.all[*aName*]
	-	*myCheckbox* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*myCheckbox* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*]
	-	*myCheckbox* = *myDocument*.getElementById(*anElementID*)
	-	*myCheckbox* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myCheckbox* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]

HTML syntax:	`<INPUT TYPE="checkbox">`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The `name` attribute of an element
	anElementID	The `ID` attribute of an element
	anItemIndex	A valid reference to an item in the collection
	aFormIndex	A reference to a particular form in the `forms` collection
Object properties:	`checked`, `defaultChecked`, `indeterminate`, `status`, `type`, `value`	
Object methods:	`handleEvent()`	
Event handlers:	`onAfterUpdate`, `onBeforeUpdate`, `onBlur`, `onClick`, `onDblClick`, `onErrorUpdate`, `onFilterChange`, `onFocus`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onRowEnter`, `onRowExit`	

Many properties, methods, and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

There isn't really a `Checkbox` object class but it is helpful when trying to understand the wide variety of input element types if we can reduce the complexity by discussing only the properties and methods of a checkbox. In actual fact, the object is represented as an item of the `Input` object class.

Checkboxes may be used in groups where each one has the same name. However, this breaks the mechanism by which a form element can be accessed associatively since there is now more than one object with the same name. The fix for this is to support an `InputArray` so that you can access the items with the same name from a collection.

Although Checkbox items should not deactivate other items in the same family in the way that Radio buttons do, you can relate their states to one another by means of the `onclick` event handler.

Unlike MSIE, Netscape does not support the `defaultValue` property or the `select()` method for this sub-class of the `Input` object.

Warnings:

- If you enumerate a `form` object that has several elements having the same name, in Netscape these will be represented by a single property of that name that refers to an `InputArray`. In MSIE, you will get multiple properties with the same name, but each will refer to a collection object. This is probably a bug in MSIE, which exhibits this behavior in version 5 for Macintosh and probably on other platforms too.
- Note that on MSIE, `Input` objects are actually `INPUT` objects because MSIE follows a general rule of naming object classes after the capitalised name of the HTML tag that instantiates them.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="RESULT">?</DIV>
<FORM onClick="handleClick()">
<INPUT TYPE="checkbox" VALUE="A" NAME="BOX_A">Selection A<BR>
<INPUT TYPE="checkbox" VALUE="B" NAME="BOX_B">Selection B<BR>
<INPUT TYPE="checkbox" VALUE="C" NAME="BOX_C">Selection C<BR>
<INPUT TYPE="checkbox" VALUE="D" NAME="BOX_D">Selection D<BR>
</FORM>
<SCRIPT>
function handleClick()
{
   myString  = "Selection [";
   myString += event.srcElement.value;
   myString += "], State [";
   myString += event.srcElement.checked;
   myString += "]";
   document.all.RESULT.innerText = myString;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Element` object, `Form.elements[]`, `FormElement` object, `Input` object, `Input.accessKey`, `onClick`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`checked`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`defaultChecked`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`indeterminate`	-	3.0 +	-	4.0 +	-	-	-	-
`status`	-	3.0 +	-	4.0 +	-	-	-	Warning
`type`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onClick`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

Checkbox.checked (Property)

The state of the checkbox is maintained in this property.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myCheckbox*.checked
HTML syntax:	`<INPUT CHECKED>`	

If the checkbox has a mark in it (depending on the UI display appearance guidelines, this may be a tick or a cross), then this value will return `true`. Otherwise it will return `false`.

Checkbox.defaultChecked (Property)

The original initial default state of a checkbox.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myCheckbox*.defaultChecked
HTML syntax:	`<INPUT CHECKED>`	

The `defaultChecked` state of an Input item is the value that was defined in the HTML document source when the page was loaded. You can use this value if you need to reset the status of a page or determine whether the user has changed the settings on an input item since the page was loaded.

Checkbox.handleEvent() (Method)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myResetButton*.handleEvent(*anEvent*)
Argument list:	*anEvent*	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event-handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`handleEvent()`

Checkbox.indeterminate (Property)

A checkbox is in this state if it was selected, but then disabled. The state cannot be accurately and unambiguously determined.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myCheckbox`*`.indeterminate`

Checkboxes may be enabled and disabled. If you have a checkbox that was enabled and then checked, and then if it is subsequently disabled, this flag property is set to `true` which indicates that the current state of the checkbox is indeterminate.

Checkbox.status (Property)

The current highlighted or checked status of the input element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myCheckbox`*`.status`

This is the current status of the checkbox item. It is either checked or not. If the checkbox has not been changed since the page was loaded from the server, then this value will be the same as the `defaultChecked` property of the checkbox.

Warnings:

- Because this is not supported on all browsers, you should use the `Checkbox.checked` property instead if portable code is important to your project.

Checkbox.type (Property)

The type value for the `<INPUT>` tag that describes the form checkbox.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myCheckbox*.type
HTML syntax:	<INPUT TYPE="CHECKBOX">	

The type value for a checkbox is always "checkbox". This value is necessary to determine the type of form element because this object is really an instance of the Input class and not the Checkbox class. There is actually no Checkbox class.

See also:	Input.type

Property attributes:

ReadOnly.

Checkbox.value (Property)

The text string for this particular checkbox object.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myCheckbox*.value
HTML syntax:	<INPUT VALUE="...">	

This value is returned to the server during a submit only if the checked state for this radio button is on.

Warnings:

- If the checkbox is used in a group or even if it isn't, the state of the checkbox is in the checked property not the value property.

See also:	Input.value

CheckerBoard() (Filter/transition)

A transition effect with the appearance of chequer board blinds opening or closing.

Availability:	JScript - 5.5 Internet Explorer - 5.5

Refer to:

filter - CheckerBoard()

ChildNodes object (Object/DOM)

A collection of all the children belonging to a DOM `Node` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myChildNodes* = *myElement*.childNodes

This is part of the internal DOM hierarchy model in the browser. There are several tree hierarchies supported and this one maintains a tree of parent-child node relationships across the document.

See also:	`Element` object, `Element.childNodes[]`, Hierarchy of objects

Chroma() (Filter/visual)

A visual filter for chroma key effects.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Filter - `Chroma()`

CITE object (Object/HTML)

An object representing the HTML content delimited by the `<CITE>` HTML tags.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myCITE* = *myDocument*.all.*anElementID*
	IE	*myCITE* = *myDocument*.all.tags("CITE")[*anIndex*]
	IE	*myCITE* = *myDocument*.all[*aName*]
	-	*myCITE* = *myDocument*.getElementById(*anElementID*)
	-	*myCITE* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myCITE* = *myDocument*.getElementsByTagName("CITE")[*anIndex*]
HTML syntax:	`<CITE> ... </CITE>`	

Argument list:	*anElementID*	The *ID* value of the element required
	anIndex	A valid reference to an item in the collection
	aName	The *name* attribute of an element

Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition – 2

This internal property returns a string value containing the class name of the containing object.

Every object type must implement this property.

It is supported by all built-in native objects in an ECMA-compliant JavaScript interpreter.

Host objects may supply any value as a `Class` identifying string. They may even masquerade as one of the built-in classes, but good sense suggests that if they do, then they must obey the protocol of that built-in class in precisely the same way as if they were a real built-in object. It's probably sensible for host implementers to avoid overloading the built-in class names like that.

At edition 2 of the ECMA standard, there is no publicly accessible method to retrieve this property in a script. However, the reserved keyword values suggest that this may be offered at a later revision of the standard.

See also:	`Array.Class`, `Boolean.Class`, `class`, `Date.Class`, `Function.Class`, Internal Property, `Number.Class`, `Object.Class`, Reserved word, `String.Class`

Property attributes:

`Internal`.

Cross-references:

ECMA 262 edition 2 – section – 8.6.2

ECMA 262 edition 3 – section – 8.6.2

class (Reserved word)

Reserved for future language enhancements.

Although you cannot request the class of a particular object, you can probably establish what class it belongs to with the `typeof` operator.

This keyword also represents a Java object type and the `class` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	`Class`, Internal Property, `java.lang.Class`, LiveConnect, Reserved word, `typeof`

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 2 – section – 11.4.3

ECMA 262 edition 3 – section – 7.5.3

ECMA 262 edition 3 – section – 11.4.3

Class method (Definition)

Methods owned by a constructor function object.

Refer to:

Static method

Class variable (Definition)

Static variables owned by a constructor function object.

See also:	Property

Refer to:

Static variable

CLASS="..." (HTML Tag Attribute)

A means of associating a tag with a stylesheet class. Represented by the `className` property of an `Element` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

In MSIE, virtually any object can be associated with a `style` object by means of the `CLASS` attribute. This is reflected into the `className` property of the object. It is especially applicable to DOM-related objects, which are considered to be sub-classed from the `Element` object. Netscape 6.0 brings that browser into line with these capabilities.

See also:	DOM, `Element` object, `Element.className`, `Element.style`, `STYLE` object (1), `style` object (2)

classes (Property)

An alternative reference to the `document.classes` property in JSS.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated – Netscape 6.0	
Property/method value type:	`Collection` object	
JavaScript syntax:	N	`classes`
	N	*`myDocument`*`.classes`

Warnings:

- This functionality is removed from Netscape 6.0.

See also:	JavaScript Style Sheets, `Document.classes[]`

CLASSPATH (Environment variable)

This is an important environment variable that helps Java code locate resources on your system. It needs to be set correctly.

See also:	Java, LiveConnect

clearInterval() (Method)

Cancel a previous `setInterval()` timer.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	- `clearInterval(`*`anIntervalID`*`)`	
	- *`myWindow`*`.clearInterval(`*`anIntervalID`*`)`	
Argument list:	*`anIntervalID`*	The ID of an interval returned by the `setInterval()` method

See also:	`Window.clearTimeout(),Window.clearInterval()`

clearTimeout() (Method)

A function that removes a pending timeout event.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	- `clearTimeout(`*`aTimeoutID`*`)`	
	- *`myWindow`*`.clearTimeout(`*`aTimeoutID`*`)`	
Argument list:	*`aTimeoutID`*	The ID of a timeout returned by the `setTimeout()` method

Warnings:

- This can cause problems if the timeout event you have identified has already been executed. Pending timeouts quite reasonably can be removed. Already executed timeout actions cannot be removed and may crash the browser if you try to remove them. Be sure that you are really removing a pending timeout.
- A better technique is to set flags in global variables and use them to inhibit the creation of a new timeout event if you are using this for a kind of interval timer.

See also:	`Window.clearTimeout()`,`Window.setTimeout()`

client object (Object/NES)

A server-side object available in NES.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	`client`
HTML syntax:	`client`	
Object methods:	`destroy()`,`expiration()`	

One `client` object is created for each browser user. It is created when the user first accesses the NES application and persists until some time after they have last visited. A timeout allows the server to garbage-collect these `session` objects and purge them out. If a client comes back again later, a new object will need to be created.

Because there is no `session` object in Netscape Enterprise Server, this object serves the purpose of maintaining session state as well as holding details of the client.

To maintain state across all session in an application, you should use the `project` object discussed in a separate topic.

`Client` objects have a limited lifetime. It is configurable but typically they will expire and be discarded after 10 minutes of inactivity.

See also:	Netscape Enterprise Server, `project` object, `response.client`, `unwatch()`,`watch()`

Method	JavaScript	JScript	NES	Notes
`destroy()`	1.1 +	-	2.0 +	-
`expiration()`	1.1 +	-	2.0 +	-

client.destroy() (Method)

This destroys the `client` object.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	`client.destroy()`

Calling this method on a `client` object will remove it and if the user makes another request, a new `client` object will have to be created.

This is slightly inconsistent with the normal way that objects are destroyed. Normally the `delete` operator would be used.

client.expiration() (Method)

This method will define the timeout after which the `client` object will expire. Used to set the life-span of a `client` object in an NES server.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	`client.expiration(`*`aTime`*`)`
Argument list:	*`aTime`*	A time value measured in seconds

This method allows the number of seconds before a session times out to be defined. After this time, the `client` object will be purged automatically and if the user connects again after that, a new `client` object will need to be created.

The expiration time can be set for individual `client` objects. You may have one that needs to persist longer based on the kind of session the user is experiencing.

For example, it might be useful to expire an e-commerce session quickly to prevent misuse. A content administrator may need a session to be active for a much longer time than usual.

Typical expiry time for this sort of thing would be about 30 minutes. This is in line with current practice of log analysis techniques and log auditing. Breaks of more than 30 minutes are considered to be multiple sessions.

Client pull techniques (Definition)

This is a technique whereby the client end pulls content from the server at regular intervals.

Refer to:

Timer events

Client-side JavaScript (Definition)

The JavaScript that gets executed in the web browser or other client application.

See also:	`.jar`, `.java`, `.js`, Desktop JavaScript, HTML file, Server-side JavaScript, Web browser

Cross-references:

Wrox *Instant JavaScript* – page – 3

Wrox *Instant JavaScript* – page – 5

clientInformation (Property)

Details of the browser, A.K.A. the `navigator` object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Navigator` object	
JavaScript syntax:	IE	`clientInformation`
	IE	*`myWindow`*`.clientInformation`

See also:	Cross platform compatibility, `Navigator` object, `Window.clientInformation`, `Window.navigator`

Property attributes:

`ReadOnly`, `DontEnum`.

Clip object (Object/Navigator)

An object that represents a clip region within a layer.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated Netscape 6.0	
JavaScript syntax:	N	*`myClip`* `=` *`myLayer`*`.clip`
	N	*`myClip`* `=` *`myStyle`*`.clip`
	N	*`myClip`* `=` *`myTextRectangle`*
	N	*`myClip`* `=` *`myRect`*
Object properties:	`bottom`, `height`, `left`, `right`, `top`, `width`	

This object represents a clipping rectangle that the visible part of a display object is viewed through. This is most likely used with a `layer` object. The layer contents would be drawn off-screen and then that part that falls within the clipping rectangle would be displayed in the window.

This can be useful for performing wipes and making parts of a layer progressively visible within some kind of transition loop.

In the MSIE browser, these rectangular objects are manufactured as needed with the `rect()` constructor function.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.clip`, `Rect` object, `style.clip`, `textRectangle` object

Property	JavaScript	JScript	N	IE	Opera	Notes
`bottom`	1.2 +	-	4.0 +	-	-	Warning, Deprecated
`height`	1.2 +	-	4.0 +	-	-	Warning, Deprecated
`left`	1.2 +	-	4.0 +	-	-	Warning, Deprecated
`right`	1.2 +	-	4.0 +	-	-	Warning, Deprecated
`top`	1.2 +	-	4.0 +	-	-	Warning, Deprecated
`width`	1.2 +	-	4.0 +	-	-	Warning, Deprecated

Clip.bottom (Property)

The bottom of a layer clip region.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated Netscape 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myClip*`.bottom`

This defines the bottom edge of the clip region. You could modify this in a loop to create a vertical downwards wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.clip.bottom`, `Rect.bottom`

Clip.height (Property)

The height of a layer clip region.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated Netscape 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myClip`*`.height`

The clip region is defined by an extent rectangle that surrounds the space occupied by the clip region. An extent rectangle is the smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.clip.height,Rect.height`

Clip.left (Property)

The left of a layer clip region.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated Netscape 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myClip`*`.left`

This defines the left edge of the clip region. You could modify this in a loop to create a horizontal wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.clip.left,Rect.left`

Clip.right (Property)

The right of a layer clip region.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated Netscape 6.0

Property/method value type:	Number primitive	
JavaScript syntax:	N	*myClip*.right

This defines the right edge of the clip region. You could modify this in a loop to create a horizontal wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	Layer.clip.right, Rect.right

Clip.top (Property)

The top of a layer clip region.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated Netscape 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	N	*myClip*.top

This defines the top edge of the clip region. You could modify this in a loop to create a vertical downwards wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	Layer.clip.top, Rect.top

Clip.width (Property)

The width of a layer clip region.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated Netscape 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	N	*myClip*.width

The clip region is defined by an extent rectangle that surrounds the space occupied by the clip region. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.clip.width`, `Rect.width`

clipboardData (Property)

A global browser variable that refers to the `clipboardData` object for the window.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`clipboardData` object	
JavaScript syntax:	IE	`clipboardData`
	IE	*`myWindow`*`.clipboardData`

See also:	`clipboardData` object, `Window.clipboardData`

clipboardData object (Object/JScript)

An object that can be used with editing operations to provide script-driven access to the clipboard contents.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*`myClipboardData`* `= clipboardData`
	IE	*`myClipboardData`* `= myWindow.clipboardData`
Object methods:	`clearData()`, `getData()`, `setData()`	

If you want to move data in and out of the clipboard on a Windows platform from within the MSIE browser, this object encapsulates the clipboard contents.

Refer to the `dataTransfer` object for a description of the `clearData()`, `getData()`, and `setData()` methods that may also be used with the `clipboardData` object.

See also:	`dataTransfer.clearData()`, `dataTransfer.getData()`, `Window.clipboardData`

Method	JavaScript	JScript	N	IE	Opera	Notes
`clearData()`	-	5.0 +	-	5.0 +	-	-
`etData()`	-	5.0 +	-	5.0 +	-	-
`setData()`	-	5.0 +	-	5.0 +	-	-

close() (Method)

A function that closes the receiving window.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`close()`
	-	*`myWindow`*`.close()`

See also:	`UniversalBrowserAccess`,`UniversalBrowserWrite`, `Window.close()`

closed (Property)

A flag indicating the window disposition.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer version – 4.0 Netscape Navigator version – 3.0 Opera browser – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`closed`
	-	`myWindow.closed`
Argument list:	`false`	The window is still open
	`true`	The window has been closed

Property attributes:

`ReadOnly`.

Refer to:

`Window.closed`

Closure object (Object/internal)

A special kind of function object that preserves prototype inheritance and scope.

Availability:	JavaScript – 1.2 Netscape – 4.0	
JavaScript syntax:	N	`myClosure = new Closure()`
Object properties:	`__parent__`, `__proto__`	

This is a special kind of object, which maintains some contextual state information when it is created.

It can behave like a function, but is a kind of function wrapper that references a function and a scope. Since it inherits everything from the `Function` object, it can behave like a function and can be called as such.

Because it also stores the scope chain at the time it is manufactured, it can restore that scope chain when it is executed.

See also:	Lexical scoping

Property	JavaScript	JScript	N	IE	Opera	Notes
`__parent__`	1.2 +	-	4.0 +	-	-	-
`__proto__`	1.2 +	-	4.0 +	-	-	-

Closure() (Object/Navigator)

A `Closure` object constructor.

Availability:	JavaScript – 1.2 Netscape – 4.0	
JavaScript syntax:	N	`new Closure(aFunction, aTarget)`
Argument list:	*aFunction*	The declaration of a function
	aTarget	An object to associate the function with

This constructor is used internally to create a `Closure` object containing the function associated with a target object.

The `Closure()` constructor is used in Netscape 4 to create an event-handler function that can be forced to run in a scope containing a target object.

Example code:

```
document.form1.myButton.onclick =
new Closure(
   function()
   {
      document.validated = false;
   },
   document.form1.myButton
   );
```

See also:	Constructor function, `constructor` property, `Global` object, Object constant

Closure.__parent__ (Property)

A reference to a scope-chain object that is preserved with the function by the `Closure` object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`ScopeChain` object	
JavaScript syntax:	N	*`myClosure`*`.__parent__`

See also:	`__parent__`, `__proto__`, Lexical scoping, `Closure` object

Closure.__proto__ (Property)

A reference to a function that is encapsulated by the `Closure` object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Function object`	
JavaScript syntax:	N	*`myClosure`*`.__proto__`

See also:	`__parent__`, `__proto__`, Lexical scoping, `Closure` object

clsid: URL (Request method)

Used by MSIE to locate ActiveX controls for the `<OBJECT>` tag.

A special request method for loading `ActiveX` objects from the locally stored object repository. This provides a portable, cross-platform, installation,independent way to refer to the folder where you have installed shared `ActiveX` objects on your system. It is more or less equivalent to the `file:` request method but without the need to specify a path to the folder.

See also:	`OBJECT.classid`, URL

Code block delimiter {} (Delimiter)

A delimiting token for a block of executable script source text.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	*aLabel*`: { `*someScript*` }`
Argument list:	*aLabel*	An optional identifier to name the code block
	someScript	Some legal JavaScript source text

A block is a list of statements that form one syntactic unit enclosed in curly brace characters (`{ }`).

This is particularly useful in conditional execution and iterative execution. Both of those are expected to operate on a single syntactic unit. A block allows that single syntactic unit to be composed of multiple lines of source script text.

Because the curly brace characters are used to delimit a block of code that comprises a list of semi-colon terminated statements, you do not need to place any semi-colons after the closing curly brace.

A block of code is most often used like this with a iterator or conditional test to either call the same section of code repetitively or to execute it as the result of a conditional expression returning a `true` value.

In compiled languages, variables declared inside a block are sometimes local to that block and are garbage-collected when the block exits. The ECMA standard indicates that variables created inside a code block will be global unless that code block is the body of a function. In ECMA-compliant interpreters, a block does not instantiate a new execution context, whereas in C language it does create a new scope within which the variables exist.

This means that variables created inside an 'if keyword' controlled compound statement will be function-local or globally accessible according to whether the 'if keyword' is in a function or global code section.

Braces must be used in pairs. Although the JavaScript interpreters may forgive you when you miss out some language elements, very subtle and difficult-to-diagnose errors can occur if you misplace a brace character.

Modern text editors give you a lot of help when balancing pairs of braces.

The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.

At version 1.2 of JavaScript, you can name the code block and use the labeled form of the `break` keyword to exit the block prematurely.

See also:	Associativity, `else if( ... ) ...`, `if( ... ) ... else ...`, Label, Operator Precedence, Punctuator

Cross-references:

ECMA 262 edition 2 – section – 12.5

ECMA 262 edition 3 – section – 12.1

Wrox *Instant JavaScript* – page – 18

CODE object (Object/HTML)

An object representing the HTML content delimited by the `<CODE>` HTML tags.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myCODE* = *myDocument*.all.*anElementID*
	IE	*myCODE* = *myDocument*.all.tags("CODE")[*anIndex*]
	IE	*myCODE* = *myDocument*.all[*aName*]
	-	*myCODE* = *myDocument*.getElementById(*anElementID*)
	-	*myCODE* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myCODE* = *myDocument*.getElementsByTagName("CODE")[*anIndex*]
HTML syntax:	`<CODE> ... </CODE>`	
Argument list:	*anElementID*	The `ID` value of the element required
	anIndex	A valid reference to an item in the collection
	aName	The `name` attribute of an element
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	

See also:	KBD object, LISTING object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

Refer to:

Element object

Code signing (Definition)

A security mechanism to allow scripts to inter-communicate.

Signed scripts are allowed higher privileges to communicate with one another because the browser assumes they are more secure.

See also:	Security policy, Signed scripts

COL object (Object/HTML)

An object that represents a <COL> HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

Inherits from:	`Element` object	
JavaScript syntax:	IE	*myCOL* = *myDocument*.all.*anElementID*
	IE	*myCOL* = *myDocument*.all.tags("COL")[*anIndex*]
	IE	*myCOL* = *myDocument*.all[*aName*]
	-	*myCOL* = *myDocument*.getElementById(*anElementID*)
	-	*myCOL* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myCOL* = *myDocument*.getElementsByTagName("COL")[*anIndex*]
HTML syntax:	`<COL> ... </COL>`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The `name` attribute of an element
	anElementID	The `ID` attribute of an element
Object properties:	`align`, `ch`, `chOff`, `span`, `vAlign`, `width`	
Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`	

This object represents the `<COL>` tag, which is used within `<TABLE>` constructs to provide a way of controlling an entire table column from a single definition. It is used in conjunction with a `<COLGROUP>` construct.

The HTML 4 specification describes functionality that currently none of the widely available browsers supports properly.

The DOM specification describes a `HTMLTableColElement` object, which is the standardized interface to this class.

See also:	`COLGROUP` object, `Element` object, `style.columnSpan`, `TABLE` object, `TABLE.rules`, `TableColElement` object, `TableColElement.align`, `TableColElement.ch`, `TableColElement.chOff`, `TableColElement.span`, `TableColElement.vAlign`, `TableColElement.width`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`ch`	1.5 +	-	6.0 +	-	-	1 +	-	-
`chOff`	1.5 +	-	6.0 +	-	-	1 +	-	-
`span`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`vAlign`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`width`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

COL.align (Property)

An attribute controlling the alignment of a column contained in a `<COL>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myCOL*.align

The alignment of the `COL` object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `char`
- `justify`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

Note that not all of these are available in every browser. In particular the `justify` and `char` values are recent additions.

See also:	`COLGROUP.align`

COL.ch (Property)

The alignment character for cells in a column arrangement.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	`myTHEAD.ch`

HTML 4.0 provides for text to be arranged in neat columns within table cells without the need to create additional tables within tables. This method of alignment is selected by setting the `ALIGN="CHAR"` HTML tag attribute. The `CHAR` HTML tag attribute is reflected in this property and is active when the `CHAROFF` HTML tag attribute is present.

See also:	`COLGROUP.ch`, `TD.ch`, `TH.ch`, `THEAD.ch`, `TR.ch`

COL.chOff (Property)

The offset of a column alignment character.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*`myTHEAD`*`.chOff`

The `CHAR` alignment style requires that an alignment character be specified with the `ch` property and that an offset measured in pixels be defined as its value. The offset value can be defined with the `CHAROFF` HTML tag attribute. The remainder of the string is offset by this distance from the alignment character.

See also:	`COLGROUP.chOff`, `TD.chOff`, `TH.chOff`, `THEAD.chOff`, `TR.chOff`

COL.span (Property)

The number of columns that the style for this object spans.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myCOL*`.span`

This corresponds to the `COLSPAN` attribute within a `<TD>` or `<TH>` table cell description. It defines how many table columns this column is to span.

See also:	`COLGROUP.span`, `TD.colSpan`, `TH.colSpan`

COL.vAlign (Property)

The vertical alignment of items within this column.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myCOL*`.vAlign`

This property controls text alignment in the vertical axis. This applies to text cells in the column group.

The `vAlign` property may be set to these values:

- `baseline`
- `bottom`
- `middle`
- `top`

See also:	`COLGROUP.vAlign`

COL.width (Property)

The width of each column in the column group.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myCOL*.width

The columns within the column group are set to a width defined by this property. Changing the value will cause the page content to be re-flowed to reflect the new value.

See also:	`COLGROUP.width`

COLGROUP object (Object/HTML)

An object that represents the `<COLGROUP>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myCOLGROUP* = *myDocument*.all.*anElementID*
	IE	*myCOLGROUP* = *myDocument*.all.tags("COLGROUP")[*anIndex*]
	IE	*myCOLGROUP* = *myDocument*.all[*aName*]
	-	*myCOLGROUP* = *myDocument*.getElementById(*anElementID*)
	-	*myCOLGROUP* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myCOLGROUP* = *myDocument*.getElementsByTagName("COLGROUP")[*anIndex*]
HTML syntax:	`<COLGROUP> ... </COLGROUP>`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The `name` attribute of an element
	anElementID	The `ID` attribute of an element
Object properties:	`align`, `ch`, `chOff`, `span`, `vAlign`, `width`	
Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`	

The <COL> and <COLGROUP> tags correspond to objects that represent the column or group of columns within a <TABLE> construct. Individual columns map to the COL object and groups of columns to the COLGROUP object. Attributes can be applied to a group of columns and overridden on an individual column basis if necessary.

The DOM specification mentions an HTMLTableColElement object, which provides the functionality of this class.

See also:	COL object, style.columnSpan, TABLE object, TABLE.rules, TableColElement object, TableColElement.align, TableColElement.ch, TableColElement.chOff, TableColElement.span, TableColElement.vAlign, TableColElement.width

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
ch	1.5 +	-	6.0 +	-	-	1 +	-	-
chOff	1.5 +	-	6.0 +	-	-	1 +	-	-
span	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
vAlign	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

Element object, Node object

COLGROUP.align (Property)

The alignment settings for a column group.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myCOLGROUP*.align

The alignment of the `COLGROUP` object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

See also:	`COL.align`

COLGROUP.ch (Property)

The alignment character for cells in a column group arrangement.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTHEAD*.ch

HTML 4.0 provides for text to be arranged in neat columns within table cells without the need to create additional tables within tables. This method of alignment is selected by setting the `ALIGN="CHAR"` HTML tag attribute. The `CHAR` HTML tag attribute is reflected in this property and is active when the `CHAROFF` HTML tag attribute is present.

See also:	COL.ch, TD.ch, TH.ch, THEAD.ch, TR.ch

COLGROUP.chOff (Property)

The offset of a column alignment character.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myTHEAD*.chOff

The CHAR alignment style requires that an alignment character be specified with the ch property and that an offset measured in pixels be defined as its value. The offset value can be defined with the CHAROFF HTML tag attribute. The remainder of the string is offset by this distance from the alignment character.

See also:	COL.chOff, TD.chOff, TH.chOff, THEAD.chOff, TR.chOff

COLGROUP.span (Property)

The span settings for a column group.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myCOLGROUP*.span

This property contains a definition of the number of adjacent columns that are affected by the <COLGROUP> tag's attributes. It corresponds to the COLSPAN attributes of the <TD> and <TH> tags and the objects that encapsulate them.

See also:	COL.span, TD.colSpan, TH.colSpan

COLGROUP.vAlign (Property)

The vertical alignment of items in a column group.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myCOLGROUP*.vAlign

This property controls text alignment in the vertical axis. This applies to text cells in the column group.

The `vAlign` property may be set to these values:

- `baseline`
- `bottom`
- `middle`
- `top`

See also:	`COL.vAlign`

COLGROUP.width (Property)

The width of items in a column group.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myCOLGROUP*.width

The columns within the column group are set to a width defined by this property. Changing the value will cause the page content to be re-flowed to reflect the new value.

See also:	`COL.width`

Collation sequence (Definition)

The order in which objects are sorted lexically according to the locale.

Collation sequences determine the way that information is sorted into the correct sequence. This is likely to be very locale-dependant and therefore if you are going to sort data, you may need to provide a means of plugging in a sort comparator function that is aware of the national language variants if your scripts belong to sites that will be deployed in foreign languages.

This is covered in some depth in the Unicode standard manual.

See also:	Localization

Collection object (Object/DOM)

An array of `Element` objects.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera browser – 3.0	
JavaScript syntax:	IE	*myCollection* = *myDocument*.all
	IE	*myCollection* = *myDocument*.children
	IE	*myCollection* = *myDocument*.filters
	-	*myCollection* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	IE	*myCollection* = *myElement*.all
	IE	*myCollection* = *myElement*.children
	IE	*myCollection* = *myElement*.filters
	-	*myCollection* = *myDocument*.getElementsByTagName(*aTag*)[*anIndex*]
Argument list:	*aName*	The `name` attribute of an element
	anIndex	A valid reference to an item in the collection
	aTag	The name of a tag
Object properties:	`length`	
Object methods:	`Item(),namedItem(),tags()`	

A `collection` object is an enhancement to the basic `array` object to provide some additional searching capabilities for managing the contents of the document object model in a web browser.

Do not confuse DOM `NodeList` arrays with `Enumerator` or `Collection` objects. The `NodeListitem()` method is subtly different from the `Enumerator.Item()` method.

See also:	TABLE.cells[]

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
length	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	1 +	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
Item()	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	1 +	Warning
namedItem()	1.5 +	5.0 +	6.0 +	5.0 +	5.0 +	1 +	-
tags()	-	3.0 +	-	4.0 +	-	-	-

Collection.Item() (Method)

Select an Element object by index number.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0
Property/method value type:	Node object
JavaScript syntax:	- *myCollection*.item(*anIndex*) - *myCollection*.item(*aSelector*) - *myCollection*.item(*aSelector*, *anIndex*) - *myCollection*[*anIndex*]
Argument list:	*anIndex* A zero based index into the collection *aSelector* A textual value that selects all matching objects

This is a search method that traverses a collection looking for an item or collection of items by the index in the collection.

If the first argument is a numeric value, the object at the indexed position is returned. You may not place a second argument in the call. This is the DOM standard specified behavior.

If the first argument is a string, then any object in the collection that has an ID or name property that matches the selector will be assembled into another collection. If there is no second argument, that new collection will be returned as a sub-set of the original receiving collection. This is an extension to the DOM specified behavior.

If the first argument is a string and the second argument is a numeric value, the sub-set collection is manufactured but the element in that collection indexed by the second argument is returned as a single object. This is also an extension to the DOM specified behavior.

This extension is useful because you can apply a filter and selection in one call without needing to extract and then store a sub-set collection. On the downside, this will repeat the sub-setting search each time it is called which can lead to performance problems.

When using the *myCollection*.item(*anIndex*) syntax variation, it is functionally equivalent to myCollection[*anIndex*].

Note that the DOM specification does not allow for the alternative array-like addressing mode, which is implemented in browsers as a convenience.

Warnings:

- You may get back a single object if there is only one item that matches. However, if the selection criteria match more than one item, you will get back an array of objects. This is slightly problematic; it would be better if you consistently got back an array even if it contained zero or only one item. You could then operate on it consistently.
- The Item() method of an MSIE Collection object is not the same as the item() method for the DOM NodeList object. The DOM specifies the method name in lower case, though it is upper case in MSIE (although JScript is somewhat forgiving of upper-lower case errors in scripts).
- In addition, the Item() method of an MSIE Collection supports several different addressing modes, whereas the item() method of a Nodelist supports only one.

See also:	Collection object, NamedNodeMap.item(), OptionsArray.item(), style.item()

Collection.length (Property)

Returns the length of a collection array.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 3.0
Property/method value type:	Number primitive
JavaScript syntax:	- *myCollection*.length

The collection behaves exactly like an array object and returns a number representing a count of all the elements in the collection.

See also:	AnchorArray.length, AppletArray.length, Arguments.length, Array.length, Attributes.length, Form.elements.length, Form.length, FormArray.length, LayerArray.length, LinkArray.length, NodeList.length, Plugin.length, PluginArray.length, ScriptArray.length, StyleSheetList.length, Window.length

Property attributes:

`ReadOnly`.

Collection.namedItem() (Method)

Select an `Element` object by `name` or `ID` value.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`Node` object	
JavaScript syntax:	- *myCollection*.namedItem(*aString*)	
	- myCollection[*aString*]	
Argument list:	*aString*	A textual value that selects all matching objects

This is a search method that traverses a collection looking for a named item or collection of items.

The argument is a string containing the `name` or `ID` value of the `Element` to be located.

Collection.tags() (Method)

Extract a sub-list of `Element` objects of a particular tag type.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Collection` object	
JavaScript syntax:	IE	*myCollection*.tags(*aTagName*)
Argument list:	*aTagName*	The name of a tag to be filtered

The collection is traversed and all objects are examined to see if they were created by an HTML tag that is the same as that specified in the argument.

The argument must always be specified in upper case and the resulting collection will contain all objects of that type selected from the receiving `collection` object.

You can then manipulate the sub-set collection in the normal way, accessing items within it by index or by other means.

See also:	`Collection` object, `Select.tags()`

Colon (:) (Delimiter)

A delimiter used with labels and conditional operators.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0

This delimiter is used with the `case` keyword and the `default` keyword in `switch` statement blocks.

Refer to the `switch` topic for details of how this is used.

See also:	`switch( ... ) ... case: ... default: ...`

Color names (Definition)

There are standard definitions of color names for use in web pages.

Property/method value type:	`String primitive`

Although the color names are used in HTML tags, you can use Hexadecimal values as well.

Here is a list of the standard color definitions:

Name:	Value:
`AliceBlue`	`#F0F8FF`
`AntiqueWhite`	`#FAEBD7`
`Aqua`	`#00FFFF`
`Aquamarine`	`#7FFFD4`
`Azure`	`#F0FFFF`
`Beige`	`#F5F5DC`
`Bisque`	`#FFE4C4`
`Black`	`#000000`
`BlanchedAlmond`	`#FFEBCD`
`Blue`	`#0000FF`
`BlueViolet`	`#8A2BE2`
`Brown`	`#A52A2A`
`BurleyWood`	`#DEB887`
`CadetBlue`	`#5F9EA0`
`Chartreuse`	`#7FFF00`
`Chocolate`	`#D2691E`
`Coral`	`#FF7F50`
`CornFlowerBlue`	`#6495ED`
`CornSilk`	`#FFF8DC`
`Crimson`	`#DC143C`

Table continued on following page

Name:	Value:
Cyan	#00FFFF
DarkBlue	#00008B
DarkCyan	#008B8B
DarkGoldenrod	#B8860B
DarkGray	#A9A9A9
DarkGreen	#006400
DarkKhaki	#BDB76B
DarkMagenta	#8B008B
DarkOliveGreen	#556B2F
DarkOrange	#FF8C00
DarkOrchid	#9932CC
DarkRed	#8B0000
DarkSalmon	#E9967A
DarkSeaGreen	#8FBC8F
DarkSlateBlue	#483D8B
DarkSlateGray	#2F4F4F
DarkTurquoise	#00CED1
DarkViolet	#9400D3
DeepPink	#FF1493
DeepSkyBlue	#00BFFF
DimGray	#696969
DodgerBlue	#1E90FF
Firebrick	#B22222
FloralWhite	#FFFAF0
ForestGreen	#228B22
Fuchsia	#FF00FF
Gainsboro	#DCDCDC
GhostWhite	#F8F8FF
Gold	#FFD700
Goldenrod	#DAA520
Gray	#808080
Green	#008000
GreenYellow	#ADFF2F
Honeydew	#F0FFF0
HotPink	#FF69B4
IndianRed	#CD5C5C
Indigo	#4B0082
Ivory	#FFFFF0
Khaki	#F0E68C
Lavender	#E6E6FA
LavenderBlush	#FFF0F5
LawnGreen	#7CFC00
LemonChiffon	#FFFACD
LightBlue	#ADD8E6

Table continued on following page

Name:	Value:
LightCoral	#F08080
LightCyan	#E0FFFF
LightGoldenrodYellow	#FAFAD2
LightGray	#D3D3D3
LightGreen	#90EE90
LightPink	#FFB6C1
LightSalmon	#FFA07A
LightSeaGreen	#20B2AA
LightSkyBlue	#87CEFA
LightSlateGray	#778899
LightSteelBlue	#B0C4DE
LightYellow	#FFFFE0
Lime	#00FF00
LimeGreen	#32CD32
Linen	#FAF0E6
Magenta	#FF00FF
Maroon	#800000
MediumAquamarine	#66CDAA
MediumBlue	#0000CD
MediumOrchid	#BA55D3
MediumPurple	#9370DB
MediumSeaGreen	#3CB371
MediumSlateBlue	#7B68EE
MediumSpringGreen	#00FA9A
MediumTurquoise	#48D1CC
MediumVioletRed	#C71585
MidnightBlue	#191970
MintCream	#F5FFFA
MistyRose	#FFE4E1
Moccasin	#FFE4B5
NavajoWhite	#FFDEAD
Navy	#000080
OldLace	#FDF5E6
Olive	#808000
OliveDrab	#6B8E23
Orange	#FFA500
OrangeRed	#FF4500
Orchid	#DA70D6
PaleGoldenrod	#EEE8AA
PaleGreen	#98FB98
PaleTurquoise	#AFEEEE
PaleVioletRed	#DB7093
PapayaWhip	#FFEFD5
PeachPuff	#FFDAB9
Peru	#CD853F

Table continued on following page

Name:	Value:
Pink	#FFC0CB
Plum	#DDA0DD
PowderBlue	#B0E0E6
Purple	#800080
Red	#FF0000
RosyBrown	#BC8F8F
RoyalBlue	#4169E1
SaddleBrown	#8B4513
Salmon	#FA8072
SandyBrown	#F4A460
SeaGreen	#2E8B57
SeaShell	#FFF5EE
Sienna	#A0522D
Silver	#C0C0C0
SkyBlue	#87CEEB
SlateBlue	#6A5ACD
SlateGray	#708090
Snow	#FFFAFA
SpringGreen	#00FF7F
SteelBlue	#4682B4
Tan	#D2B48C
Teal	#008080
Thistle	#D8BFD8
Tomato	#FF6347
Turquoise	#40E0D0
Violet	#EE82EE
Wheat	#F5DEB3
White	#FFFFFF
WhiteSmoke	#F5F5F5
Yellow	#FFFF00
YellowGreen	#9ACD32

If you prefer to compute the color names, there is a very neat color picker tool on the Netscape developer's web site.

See also:	Color value

Web-references:

http://home.netscape.com/computing/webbuilding/studio/feature19981111-5.html

Color value (Advice)

Color values can be specified numerically or mnemonically.

Property/method value type:	`String primitive`

A numeric color specification is a 6 digit (3 pairs) hexadecimal value with a leading hash symbol.

Each pair of digits defines the intensity of a single color (RGB) in the display.

The web-safe palette of 216 colors is defined by every possible combination of the following values:

00, 33, 66, 99, CC and FF

There are other values supported through the mnemonic named color palette, which includes 100 shades of gray.

You can use non-web-safe values if you have a greater than 8 bit deep display. These days most platforms can resolve at least 16 bits, but some legacy systems may have problems resolving non-web-safe colors.

When using color values for defined styled colors, the `rgb()` function can be used.

In the MSIE browser, the values can also be specified as 32 bit integer values, although in practice this is extremely difficult to manage.

In certain circumstances, where an alpha value is available, the color value can be specified using hexadecimal notation to define a 32 bit integer. In this case, the value is comprised of four pairs of hex digits as follows:

0xAARRGGBB

The value AA controls the alpha channel transparency, while the RR, GG, and BB values are the intensity of Red, Green and Blue respectively.

When passed as a string value, the same hexadecimal value can be used but must be preceded by a hash rather than the `0x` prefix. Thus `0xAARRGGBB` becomes `#AARRGGBB`

Warnings:

- Be wary of spellings when specifying color values. The UK English spelling of "colour" will not set the property values you intend. Property values to do with color values will be spelled "color". In addition some of the color names may be spelled in the American English tradition.

See also:	Colour names, `rgb()`

Comma expression (Definition)

Used to separate individual operands or arguments.

Availability:	ECMAScript edition – 2

Comma expressions occur rarely and are used to evaluate several expressions at once.

See also:	Comma operator (,), Expression

Cross-references:

ECMA 262 edition 2 – section – 11.14

ECMA 262 edition 3 – section – 11.14

Comma operator (,) (Delimiter)

An argument separator token.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

The comma operator provides a way to evaluate several assignment expressions at once.

It is provided as a way to express the arguments to a function in a manner consistent with the ECMA standard.

It is also used in variable declaration lists to declare more than one variable with a single `var` statement.

Its most subtle use is to exploit the side effects of an expression without assigning the result of that expression to anything. This is actually so subtle as to confuse what it's actually doing. There is probably a simpler way to express the same functionality that doesn't execute any slower and is a lot easier to maintain.

The associativity is left to right. This means that the expressions should be executed in left to right order of appearance in the script source text. The result of the entire expression will be the evaluation of the last comma-separated item. It is not good style to use a comma separated list as an RValue in an assignment to an LValue.

Example code:

```
// Declaring several variables
var e, c, d, x, y;

// Arguments in a for iterator header
for(a=0, b=0; a<10 ; a++, b++)
{
;
}

// assign variables
c = 1;
d = 2;
x = 3;
y = 4;
z = 5;

// Add c to d and assign the result to e with incrementing
// side effects
e = (x++, y++, c) + (z--, d);
```

See also:	Associativity, Comma expression, `Document.write()`, `Document.writeln()`, Operator Precedence, `var`

Cross-references:

ECMA 262 edition 2 – section – 11.14

ECMA 262 edition 3 – section – 11.14

Wrox *Instant JavaScript* – page – 21

Comment (Definition)

Sections of inactive code.

Availability:	ECMAScript edition – 2

A comment is some text that is embedded within the script source text but is ignored by the interpreter. It is intended to provide guidance and documentation to the reader of the source code.

Comments can be contained completely on a single line or can span multiple lines.

A single-line comment can contain any character except for a line terminator. The line terminator at the end of the single line comment is not part of the comment and must obey the rules of placement of line terminators in general. Multiple-line comments are replaced by a single line terminator regardless of how many line terminators they actually contain. This means that a multiple-line comment behaves syntactically as if it were a line terminator.

Warnings:

- You cannot nest multi-line comments within one another.

See also:	Comment (// and /* ... */), Escape sequence (\), Lexical convention, Lexical element, Line, Multi-line comment, Script Source Text, Single line comment

Cross-references:

ECMA 262 edition 2 – section – 6

ECMA 262 edition 2 – section – 7.3

ECMA 262 edition 2 – section – 7.8.2

ECMA 262 edition 3 – section – 6

ECMA 262 edition 3 – section – 7.4

O'Reilly *JavaScript Definitive Guide* – page – 29

Wrox *Instant JavaScript* – page – 17

Comment (// and /* ... */) (Delimiter)

Mark a multi-line comment block.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
JavaScript syntax:	- /* First line of comment text - // Comment text - Second line of comment text - Third line of comment text */

There are two kinds of comment blocks supported by JavaScript. They each have a different kind of delimiter.

Single-line comments commence with a pair of slash characters (//) and stop at the end of the current line.

Multi-line comments start with a slash-asterisk (/*) and finish at the first following asterisk-slash (*/). This means you cannot nest multiple line comment blocks inside one another. The disadvantage with that is that it is common practice in some languages to comment out sections of code by marking the start and end of the inactive sections as a multi-line comment block. This won't work if there are any multi-line comment blocks anywhere in this inactive block. A better technique, although slightly more cumbersome is to use single line comments (//) to 'switch-off' the lines of code you want to deactivate. Thus:

```
//a = 1000;
//b = 2000;
//c = 3000;
```

See also:	Comment, Lexical convention, Line, Line terminator, Multi-line comment, Single line comment

Cross-references:

ECMA 262 edition 2 – section – 7.3

ECMA 262 edition 3 – section – 7.4

COMMENT object (Object/DOM)

An object that represents a section of HTML enclosed in comment delimiter tags.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`CharacterData` object	
JavaScript syntax:	IE	*myCOMMENT* = *myDocument*.all.*anElementID*
	IE	*myCOMMENT* = *myDocument*.all.tags("COMMENT")[*anIndex*]
	IE	*myCOMMENT* = *myDocument*.all[*aName*]
	-	*myCOMMENT* = *myDocument*.getElementById(*anElementID*)
	-	*myCOMMENT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	N	*myCOMMENT* = *myDocument*.createComment(*aString*)
	-	*myCOMMENT* = *myDocument*.getElementsByTagName("COMMENT")[*anIndex*]
HTML syntax:	`<!-- ... -->`	
Argument list:	*anIndex*	An item within the collection
	aString	A comment string
	aName	The `name` of an element
	anElementID	The `ID` attribute of an element
Object properties:	`text`	
Object methods:	`click(),getAttribute(),removeAttribute(),setAttribute()`	
Collections:	`all[],children[]`	

It is somewhat unlikely you would ever want to modify the contents of a comment tag. However, access to the text contained within it may be a way of passing hidden data values to your scripts without them being visible in the displayed page. Of course they would still be visible in the document source, but you might be able to avoid the creation of a `<FORM>` and hidden `<INPUT>` object.

Warnings:

- The DOM level 1 specification describes this as a `Comment` object, but MSIE implements it as a `COMMENT` object instead. You may need to be aware of this in case other platforms implement the DOM specified class exactly as it is intended.

See also:	`CharacterData` object, `Document.createComment()`, `Element` object, Hiding scripts from old browsers

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`text`	-	3.0 +	-	4.0 +	-	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`click()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	Warning
`getAttribute()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	Warning
`removeAttribute()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	Warning
`setAttribute()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	Warning

Inheritance chain:

`CharacterData` object, `Node` object

COMMENT.text (Property)

The text within a comment block.

Availability:	JScript – 3.0 Internet Explorer version – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myCOMMENT`*`.text`

This is the internal text within the comment block. The DOM standard treats comment blocks as character data to standardize the access to their content, but you should not expect to be able to access the internal text of a comment for any functional purpose.

Compatibility (Definition)

Creating web content is an exercise in portable programming that would severely test the most experienced programmer.

There are so many issues to do with portability and compatibility that unless you are doing something very simple, its is an amazing achievement to get your web page to display even remotely similarly on several browsers or platforms.

Leaving aside the creative issues to do with color matching, gamma correction, and display resolution, there are problems with fonts, style sheets and table layout. Layers and dynamic HTML behave differently even within the same browser family as the browsers are revised. Even scrolling a page means you have to increment the scroll value in opposite directions in Netscape and MSIE, and Netscape won't scroll at all without the scroll bars being visible.

To cope with this, it helps to break the compatibility issues down into sub-sets and to understand the underlying reason for incompatibility. You do need to know about and be able to identify issues based on platforms, browsers, or versions of client software.

See also:	`<NOSCRIPT>`, `<SCRIPT LANGUAGE="...">`, Browser version compatibility, Compatibility strategies, Cross browser compatibility, Cross platform compatibility, Defensive coding, `Document`, Plugin compatibility issues, Portability, Server-side browser detection

Cross-references:

Wrox *Instant JavaScript* – page – 60

Compatibility strategies (Advice)

Choose how compatible you want to be from the outset.

You need to establish a strategy for how compatible you need to be. Here are some questions you need to think about and consider their relative importance.

- Is it important that your pages work perfectly in browsers that do not support JavaScript? If so, don't deploy any at all.
- If you really need JavaScript, can you design your forms and page content to degrade gracefully? If not, you may need to browser-detect and present warnings. If you cannot browser-detect with JavaScript at the client end, you may need to browser-detect at the server end and return a different page.
- Are you in control of browser deployment to the target desktop systems? This is likely in an intranet situation. If it is true, then you can use all of the features of the browser your deploy, including platform-specific and version-specific capabilities. You may need to co-ordinate content upgrades with browser upgrades though.
- Are you only using JavaScript to define proxy settings? If so, you only need to be cross-platform and browser-version aware.
- Are you only using JavaScript to set preference values in Netscape? If so, you only need to be cross-platform and browser-version aware.
- Can you sacrifice old-browser users? If you can get some meaningful statistics of your readership, you may find that only a few percent have an old browser. Are you prepared to support all users or can you disregard some? You need to set a threshold, say 5%, and work out a sub-set of browsers that you will support and from that derive a functionality profile that your must enforce in your design department.
- Must you guarantee to work on all browsers of every vintage? In which case, you could consider the lowest common denominator approach and only use those capabilities that are supported by your choice of target browsers.

- Do you think you need to only support the latest version of a browser? Users are generally keen to upgrade if it is well worth their while. They likely may not consider your site a compelling reason to upgrade though.
- Are you hardwired to a very specific platform/browser combination? Some sites only work on MSIE running on Windows. This is very frustrating for Netscape users or people using any web browser on other non-Windows platforms. Things change and you cannot rely on Microsoft Windows always being the dominant platform. This could change within the hardware replacement cycle that large corporations like to implement. That might be between 2 and 4 years. If your site only works on a single platform, your traffic may go down as that platform loses market share.
- Do you just not care about it? If your page breaks on someone's browser, so what? Maybe it's not important at all. However, don't expect that user to come back – ever. If the script fails to draw some cute animation or change a color the effect is likely to be cosmetic anyway and may not even be noticed. Maybe it fails fairly gracefully if you are lucky.

See also:	Compatibility, `Date` object

Completion type (Definition)

An internal type used by the interpreter.

Availability:	ECMAScript edition – 2

This is an internal type used by the interpreter for processing expression evaluation results. It cannot be stored as an object property.

A completion happens when a statement is evaluated and completes normally. A completion can involve a value and may still be a normal completion.

Abrupt completions are triggered by `break`, `continue`, and `return` statements, which are all classified as program steps that may redirect the flow of control. This may cause sections of code to be skipped or repeated. In the case of a `return`, it may prematurely exit a function.

In the set of ECMA-defined reserved words, there is evidence that new flow control changes may be introduced and therefore other completion type values may be added later.

The set of completion type values are summarized in this table:

Type:	Reason:
Normal completion	-
Normal completion after evaluation	-
Abrupt completion	`break`
Abrupt completion after evaluation	`break`
Abrupt completion	`continue`
Abrupt completion after evaluation	`continue`
Abrupt completion	`return`
Abrupt completion after evaluation	`return`

See also:	`break`, `continue`, Identifier resolution, `return`, Scope chain, Type

Cross-references:

ECMA 262 edition 2 – section – 8.9

ECMA 262 edition 3 – section – 8.9

Compliance (Overview)

The degree to which an implementation adheres to the standard.

The formal definition of the core JavaScript language is embodied in the ECMA standard 262.

Until recently, where implementations conformed it was to the second edition of the standard, which was published in August 1998.

In December 1999, the third edition was made available and new implementations or revisions to existing products released in the early part of the year 2000 onwards complied with that new version of the standard.

To comply with the standard, an implementation must provide and support all the types, values, objects, properties, methods, and program syntax described in the standard.

See also:	Behavior, Conformance, Definition, ECMA, ECMAScript, ECMAScript – edition 2, ECMAScript – edition 3, Limits

Compositor() (Filter/visual)

As content is added to an object, it can be colored to indicate it is changed content.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

Filter - `Compositor()`

Compound statement (Definition)

A block of code handled as if it were one statement.

A compound statement is a block of one or more statements gathered together and executed as if they were a single statement. This is a somewhat wordy description of what a function declaration provides. Other similar uses are the code that is iterated in a `while` or `for` loop and the code that is conditionally executed in an `if ... else` construct.

Any variables created within a compound statement will be local to the enclosing function if the compound statement exists inside a function's execution context, otherwise they will be globally scoped. In JavaScript you cannot localize the scope of a variable to within an `if()`, `while()`, or `for()` construct in the way that you can in C language.

See also:	Definition, `for( ... in ... ) ...`, Function code, `if( ... ) ...`, `if( ... ) ... else ...`, Scope chain, Statement, `var`, `while( ... ) ...`

Concatenate (+) (Operator/string)

Join two strings end to end. See Addition for numeric values.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0
Property/method value type:	`String primitive`

See also:	Add (+), Additive operator, `Array.join()`, `Array.toString()`, Concatenate then assign (+=), String, String, String literal, `String` object, String operator, `String.concat()`, `String.split()`, Type conversion, String concatenate (+)

Cross-references:

ECMA 262 edition 2 – section – 11.6.1

Wrox *Instant JavaScript* – page – 21

Concatenate then assign (+=) (Operator/assignment)

Concatenate two string operands and assign the result to the first. See Addition for numeric values.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0
Property/method value type:	`Number primitive`

JavaScript syntax:	-	*anOperand1* += *anOperand2*
Argument list:	*anOperand1*	A numeric value that can be assigned to
	anOperand2	Another numeric value

Concatenate the right operand to the left operand and assign the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 + anOperand2;
```

Although this is classified as an assignment operator, it is really a compound of an assignment and a concatenation operator.

It also works with numeric values and will add the second to the first.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

The new value of *anOperand1* is returned as a result of the expression.

Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Add then assign (+=), Assign value (=), Assignment expression, Assignment operator, Associativity, LValue, Operator Precedence

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

Conditional code block (Pre-processor)

A pseudo pre-processor mechanism for conditionally executing code in MSIE.

There is an implementation of a C language inspired pre-processor in the MSIE JScript interpreter.

The usual pre-processor directives for conditional code use are reproduced here except that in the C language they are prefixed with a hash symbol (`#`) and in JScript the commercial at sign (`@`) is used instead.

There are several directives and a set of pre-defined constants

- `@cc_on` Statement
- `@if` Statement

- `@set` Statement
- `@elif( ... ) ...` Statement
- `@else ...` Statement
- `@end` Statement
- `@<variable_name>` Reference
- `@_alpha` Pre-defined constant
- `@_jscript` Pre-defined constant
- `@_jscript_build` Pre-defined constant
- `@_jscript_version` Pre-defined constant
- `@_mac` Pre-defined constant
- `@_mc680x0` Pre-defined constant
- `@_PowerPC` Pre-defined constant
- `@_win16` Pre-defined constant
- `@_win32` Pre-defined constant
- `@_x86` Pre-defined constant

Warnings:

- This is not supported prior to MSIE version 4.

See also:	Pre-processing - `@cc_on`, Pre-processing - `@elif( ... ) ...`, Pre-processing - `@else ...`, Pre-processing - `@end`, Pre-processing - `@if( ... ) ...`

Conditional comment (HTML Tag)

A portability trick that only works in Netscape.

Availability:	JavaScript – 1.2 Netscape – 4.0

Conditional comments use JavaScript entities to enclose a block of JavaScript and only execute it conditionally on some value being `true`.

It is accomplished by embedding a JavaScript entity with a logical expression evaluation in it. If the expression proves `true` then the `<SCRIPT>` HTML tag enclosed in the comment block is parsed, otherwise it is ignored.

This is how it's done. A conditional comment is formed by adding an ampersand character to the leading tag of a comment. Rather than use "`<!--`" the comment is introduced with "`<!--&`" instead. The comment is closed in the normal way with a trailing "`-->`" string. Inside the comment a `<SCRIPT></SCRIPT>` block is placed with some global code to be executed if called for.

Warnings:

- This only works in Netscape 4 or later, which limits its usefulness somewhat. MSIE supports an alternative, but completely incompatible technique that only works inside the `<SCRIPT>` tag.

Example code:

```
<!--&{navigator.userAgent == "Mozilla/4.7 (Macintosh; I; PPC) "};
<SCRIPT>
document.write("Power Macintosh running Navigator 4.7");
</SCRIPT>
-->
```

See also:	Adding JavaScript to HTML

Conditional expression (Definition)

Conditionally execute one code branch or another.

Availability:	ECMAScript edition – 2

Conditional expressions test a logical expression and perform one of two possible alternative code blocks.

The grammar for a conditional expression in ECMA-compliant JavaScript implementation is slightly different from that you may have seen before in Java and C language. Java and C set restrictions on the kind of expression you can put into the second code block, whereas JavaScript does not. This is a subtle distinction and is intended to simplify the use of this expression and avoid the use of a comma operator.

See also:	Conditionally execute (`?:`), Expression

Cross-references:

ECMA 262 edition 2 – section – 11.12

ECMA 262 edition 3 – section – 11.12

Wrox *Instant JavaScript* – page – 18

Conditional operator (Definition)

Conditionally execute one code branch or another.

Availability:	ECMAScript edition – 2

Refer to:

Conditionally execute (`?:`)

Cross-references:

ECMA 262 edition 2 – section – 11.12

ECMA 262 edition 3 – section – 11.12

Conditionally execute (?:) (Operator/conditional)

Conditionally execute one code branch or another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	Depends on arguments	
JavaScript syntax:	- *aCondition* ? *someCode* : *moreCode*	
Argument list:	*aCondition*	A relational or logical expression that yields `true` or `false`
	moreCode	Code that is executed if *aCondition* is `false`
	someCode	Code that is executed if *aCondition* is `true`

The two associated code blocks are executed according to the value yielded by a Boolean test on the first operand. If it is `true`, then the first code block is executed, otherwise the second is used.

The associativity is right to left.

Refer to the Operator Precedence topic for details of execution order.

This is sometimes called a Ternary operator, because it takes three operands.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
mySwitch = false;
myResult = (mySwitch) ? "TRUE VALUE" : "FALSE VALUE" ;
document.write(myResult);
</SCRIPT>
</BODY>
</HTML>
```

See also:	Associativity, Conditional expression, Flow control, `if( ... ) ...`,`if( ... ) ... else ...`, Operator Precedence, Selection statement, Ternary operator

Cross-references:

ECMA 262 edition 2 – section – 11.12

ECMA 262 edition 3 – section – 11.12

config.jsc (Special file)

A JavaScript configuration file for Netscape.

Refer to:

Preferences

Cross-references:

Wrox *Instant JavaScript* – page – 59

confirm() (Method)

A dialog box to get confirmation from the user.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *myResult* = confirm(*aString*) - *myResult* = *myWindow*.confirm(*aString*)
Argument list:	*aString* — Some text to explain what is to be confirmed

See also:	`Window.alert(),Window.confirm()`

Conformance (Definition)

An interpreter may or may not conform to the ECMAScript specification.

Availability:	ECMAScript edition – 2

Conforming implementations of ECMA 262 edition 2 must properly support Unicode version 2.0 and ISO/IEC 10646-1 with UCS-2 as the encoding form. There are various sub-clauses to that encoding requirement and implementers should build their systems around the specifications laid down therein. Edition 3 requires slightly different character coding support.

Conforming implementations may provide additional capabilities. In particular, functionality flagged under the 'Future Reserved Word' category is encouraged by ECMA.

A strictly conforming implementation is one that only provides the features outlined in the standard. Since ECMAScript only defines core functionality and absolutely none of the host environment, a strictly conforming implementation of ECMAScript would be of limited use.

A conforming implementation provides the behavior outlined in the standard with some allowable additional behavior to support the hosting environment.

See also:	Compliance, ECMA, ECMAScript, ECMAScript – edition 2, ECMAScript – edition 3

Cross-references:

ECMA 262 edition 2 – section – 2

ECMA 262 edition 3 – section – 2

Connection object (Object/NES)

An object that represents a connection from the server to the back-end database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
JavaScript syntax:	NES	`myConnection = myDbPool.connection(aName, aTimeout)`
Argument list:	`aName`	A connection name
	`aTimeout`	Timeout in seconds
Object properties:	`prototype`	
Object methods:	`beginTransaction(),commitTransaction(),connected(), cursor(),execute(),majorErrorCode(),majorErrorMessage(), minorErrorCode(),minorErrorMessage(),release(), rollbackTransaction(),SQLTable(),storedProc(),toString()`	

This object is used to maintain the connection state details between the Netscape Enterprise Server and the backend database it is retrieving data from.

A `connection` object is created by calling the `connection()` method of the `DbPool` object.

Example code:

```
<SERVER>
// An example of how to create a connection object
// Based on the one in Wrox Professional JavaScript
myDbPool     = new DbPool("ODBC", "myDatabase", "", "", "");
myConnection = myDbPool.connection("ExampleConnection", 30);
myConnection.SQLTable("SELECT * FROM MY_TABLE");
</SERVER>
```

See also:	`DbPool.connection()`, Netscape Enterprise Server, `unwatch()`, `watch()`

Property	JavaScript	JScript	NES	Notes
`prototype`	1.2 +	-	3.0 +	-

Method	JavaScript	JScript	NES	Notes
`beginTransaction()`	1.2 +	-	3.0 +	-
`commitTransaction()`	1.2 +	-	3.0 +	-
`connected()`	1.2 +	-	3.0 +	-
`cursor()`	1.2 +	-	3.0 +	-
`execute()`	1.2 +	-	3.0 +	-
`majorErrorCode()`	1.2 +	-	3.0 +	-
`majorErrorMessage()`	1.2 +	-	3.0 +	-
`minorErrorCode()`	1.2 +	-	3.0 +	-
`minorErrorMessage()`	1.2 +	-	3.0 +	-
`release()`	1.2 +	-	3.0 +	-
`rollbackTransaction()`	1.2 +	-	3.0 +	-
`SQLTable()`	1.2 +	-	3.0 +	-
`storedProc()`	1.2 +	-	3.0 +	Warning
`toString()`	1.2 +	-	3.0 +	-

Connection.beginTransaction() (Method)

Commence a new transaction with the database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
JavaScript syntax:	NES	*myConnection*`.beginTransaction()`

Call this method to signify the start of a new transaction with the database. This is a reference point to which you can rollback the changes if necessary. Note that you cannot rollback after the commit has occurred.

Connection.commitTransaction() (Method)

Commit the changes made to the database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
JavaScript syntax:	NES	*myConnection*.commitTransaction()

Relational databases support the facility of a two phase commit. That is, any changes made to the database require that a commit is requested before the session is disconnected. If the session is disconnected without a commit, then the changes are unwound and discarded leaving the database in the state it was in after the previous commit.

Sometimes people place a commit after every transaction. This is somewhat wasteful and can lead to low performance. Placing a commit every 25 or so transactions is better.

You do need to be sure that you have made the changes you need to and that any referential integrity is maintained and any constraints on the data are satisfied, otherwise the commit may fail.

Connection.connected() (Method)

A method that returns a flag indicating the state of the connection.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	NES	*myConnection*.connected()

If the session is currently connected to the database, this method will return a `true` value. If the session is no longer connected it yields `false`.

Connection.cursor() (Method)

Create a cursor object on the connection with the SQL database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	Cursor object	
JavaScript syntax:	NES	*myConnection*.cursor(*aQuery*)
	NES	*myConnection*.cursor(*aQuery*, *aFlag*)
Argument list:	*aQuery*	A valid SQL query for the database
	aFlag	Indicates whether the cursor can be updated

These `Cursor` objects are used to run SQL queries against the database we are currently connected to. It is probably a good idea to consult the reference documentation for your database to fully understand how it handles cursors.

See also:	`Cursor` object

Connection.execute() (Method)

Execute some SQL on the database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*myConnection*`.execute(`*someSQL*`)`
Argument list:	*someSQL*	A string containing valid SQL for the database

The SQL to be executed on the database is passed as an argument to this method.

Connection.majorErrorCode() (Method)

Provide the code for an error raised by the database server or the ODBC interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*myConnection*`.majorErrorCode()`

For a status code value of 5 when using the Oracle database, this yields a return code from the Oracle Call-level interface.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the SQL server message number.

For a status code value of 7 when using the Informix database, this yields the Informix error code.

For a status code value of 7 when using the Sybase database, this yields the DB-Library error number.

See also:	`database.majorErrorCode()`, `DbPool.majorErrorCode()`, Error handling, Status code

Connection.majorErrorMessage() (Method)

Provide the descriptive text message for an error raised by the database server or by the ODBC interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myConnection`*`.majorErrorMessage()`

For a status code value of 5 when using the Oracle database, this yields a text string describing the server error.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields a text string from SQL server.

For a status code value of 7 when using the Informix database, this yields the text string from the vendor error library.

For a status code value of 7 when using the Sybase database, this yields a text string from the DB-Library.

See also:	`database.majorErrorMessage()`, `DbPool.majorErrorMessage()`, Error handling, Status code

Connection.minorErrorCode() (Method)

Returns a supplementary error code for an error raised by the database server or the ODBC interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myConnection`*`.minorErrorCode()`

For a status code value of 5 when using the Oracle database, this yields an operating system error code from the Oracle Call-level interface.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the severity level from SQL server.

For a status code value of 7 when using the Informix database, this yields the ISAM error code.

For a status code value of 7 when using the Sybase database, this yields the severity level of the error from the DB-Library.

See also:	`database.minorErrorCode()`, `DbPool.minorErrorCode()`, Error handling, Status code

Connection.minorErrorMessage() (Method)

Returns a supplementary error message text for an error raised by the database server or the ODBC interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*myConnection*`.minorErrorMessage()`

For a status code value of 5 when using the Oracle database, this yields the Oracle server name.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the SQL server name.

For a status code value of 7 when using the Informix database, this yields a text string describing the ISAM error.

For a status code value of 7 when using the Sybase database, this yields the text of the operating system error from the DB-Library.

See also:	`database.minorErrorMessage()`, `DbPool.minorErrorMessage()`, Error handling, Status code

Connection.prototype (Property)

The prototype for the `connection` object that can be used to extend the interface for all `connection` objects.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`Connection` object	
JavaScript syntax:	NES	`Connection.prototype`
	NES	*myConnection*`.constructor.prototype`

Refer to:

`prototype` property

Connection.release() (Method)

Releases this connection back into the pool for reuse.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
JavaScript syntax:	NES	*myConnection*`.release()`

There are a limited number of connections available for accessing the database. This is intentional. It prevents the database from being swamped by connections.

The limit may be large or small depending on the scale of the database server and the software it is using.

Requesting a connection from the `DbPool` object consumes one of the available connections. You must ensure that you release them back to the pool when you are finished, otherwise your server will soon use them all up and will be unable to form new connections to the database.

Connection.rollbackTransaction() (Method)

A means of undoing transactions that have not yet been committed.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
JavaScript syntax:	NES	*myConnection*.rollbackTransaction()

Until you commit a transaction on the database, you can rollback to the previous commit state. Once you have committed a transaction, this opportunity to undo is lost. To undo after that, you must make changes under your own script's control. That means that if you anticipate that possibility, you will need to remember the previous values. On the whole, it's easier to let the database do this and to commit only when you are sure the transaction is complete and correct.

Connection.SQLTable() (Method)

Create an HTML table as a result of a SQL query on the database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	String primitive	
JavaScript syntax:	NES	*myConnection*.SQLTable(*someSQL*)
Argument list:	*someSQL*	A string containing valid SQL for the database

The argument to this method is a database query that is expected to yield some rows and columns. These are then reformatted with the requisite HTML tags to form a table.

The table is likely to be fairly generic in appearance so you may need to modify the HTML that is returned or perhaps you could use style sheets to control its appearance.

Connection.storedProc() (Method)

This method creates a `Stproc` object and then runs the stored procedure on the database server.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0

JavaScript syntax:	NES	*myConnection*.storedProc(*aProcName*)
	NES	*myConnection*.storedProc(*aProcName*, *aProcParm*)
Argument list:	*aProcName*	The name of a stored procedure to call
	aProcParm	A parameter or parameters to pass to the stored procedure

Stored procedures are pre-programmed code that lives inside the database. They generally encompass searches that are more complex than the simple `select … from … where …` SQL statements.

Stored procedures in some database products cannot yield a record structure the same as a select query would. This is because they are not associated with any particular table and so they return the results as a resultset rather than a table record. You may need to do additional work to turn the resultset into the objects you need.

Warnings:

- Every middle-ware application server supports a completely different way of running stored procedures. At least it seems like that sometimes when you are using several different products. Fundamentally they all do the same thing but databases provide different mechanical implementations and so there are many different ways in which to develop code to exploit them.

See also:	`Stproc` object

Connection.toString() (Method)

Returns a string containing a representation of the `Connection` object.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*myConnection*.toString()

The value of the object is converted to a string value that represents its value.

const (Reserved word)

Reserved for future language enhancements.

This keyword suggests that future standardization may support immutable constant values. This may allow stronger type casting of formal parameters in function prototypes.

See also:	Function prototype, Reserved word, Type, volatile

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Constant (Definition)

A literal description of a fixed value.

A constant is a lexical element that represents a set numerical or string value.

Numeric values can be integer or floating-point. String values may define only a single character but are nevertheless still considered to be a string.

The `Number` and `Math` objects provide constant values as properties of the object class. These are static constants. Other static constants are provided by the implementation. For example the `Event` object supplies a set of masks in some implementations. These masks can be used to determine keyboard states. The DOM standard defines others as part of the event model.

See also:	Character constant, Constant expression, Escape sequence (`\`), Floating constant, Integer constant, Lexical element, Literal, `Math` object, `Number` object, Primary expression, Variable

Cross-references:

Wrox *Instant JavaScript* – page – 13

Constant expression (Definition)

A constant expression is a combination of constants and an operator.

A constant expression can always be degenerated to a simpler constant form.

For example `100 + 101` is a constant expression, but could be replaced by the value `202`.

A constant expression involving strings is `"abcdef" + "ABCDEF"`. This yields the result `"abcdefABCDEF"`.

Warnings:

- In JavaScript you should always try to avoid the use of string concatenation unless you really have to use it. It is very useful. However, in Netscape and MSIE, over-using a string concatenation in a loop can lead to significant memory leaks. It is quite easy to leak several megabytes of memory in a few minutes simply by concatenating strings and passing them to a `document.write()`. The garbage does eventually get collected, but not until the page is refreshed. This means that a page that implements a ticker for instance can leak horribly until the page is refreshed. There are techniques you can employ to minimize this effect and you could trigger a garbage collection by reloading the page periodically under script control.

See also:	Arithmetic constant, Character constant, Constant, Date constant, Escape sequence (`\`), Expression, Expression statement, Floating constant, Floating-point constant, Initialization, Integer constant, JavaScript language, Logical constant, Object constant

Constraint (Definition)

A restriction placed on a script to be executed.

A constraint determines a restriction that the syntax and semantics of the language must set upon the interpretation of the elements of the language. Violating these constraints should generate an interpretation error in a compliant implementation.

See also:	Definition, Diagnostic message, Error handling

Construct (Property/internal)

An object constructor call.

Availability:	ECMAScript edition – 2

The internal constructor is invoked via the `new` operator.

This is not implemented by all objects. Those that do support it are called constructor objects. In other languages these might be called factory objects.

Warnings:

- The `global` object does not have a `Construct` property and you cannot make copies of it with the `new` operator.

See also:	Constructor function, `constructor` property, Internal Method

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 – section – 15

ECMA 262 edition 3 – section – 15.1.4

Constructor function (Definition)

A function that can create new objects.

Availability:	ECMAScript edition – 2

A constructor is analogous to a factory class in a truly object-oriented system. It instantiates new objects of its class by copying a built-in prototype object.

A constructor function is a method that creates and initializes new objects or values. It is a way of calling the `constructor` object as a function rather than with the `new` operator.

These constructor functions are available from the `global` object; the host implementation may add others for you to use:

- `Object()`
- `Function()`
- `Array()`
- `String()`
- `Boolean()`
- `Number()`
- `Date()`

Calling the constructor as a function is a way of carrying out type conversion.

You can also create your own constructor functions. You name them with the name of the class you want to create. Depending on how you implement them, they may or may not work as type converting functions.

When we use the `new` operator, it understands that the function is a constructor and as it constructs the new object, it associates the constructor function with it so you can locate it again, via the `prototype` and `constructor` properties. Because this function makes use of the '`this`' keyword, it could be a method belonging to another object. However, it doesn't have to belong to an object to be used as a constructor in the first place. The object creation process will properly associate it in due course when it needs to.

You don't need to pass parameters to add properties to objects as they are created.

Constructors are generally a better way of making self documenting objects than simply instantiating more copies of the `Object` object. In addition, there is more opportunity to reuse and share code between multiple instances. It's probably not a good thing to add properties and methods to the base `Object` class as it means they would get inherited just about everywhere, because ultimately all prototype inheritance chains descend from the topmost `Object`.

With a constructor, you can simulate arrays by making them from objects and property components. This may be useful if you want to run an array-based script in a very old JavaScript implementation although these days that likelihood of that is diminishing rapidly.

The Netscape browser creates a constructor for virtually every object it instantiates. This can be an aid to debugging and making more flexible scripts. You can inspect an object by requesting its `constructor` property. This will normally convert to a string that contains the function definition and to determine the class it may be more useful to request the `constructor.name` property instead.

This technique is not so useful in MSIE where constructors are only made available for objects that can genuinely be instantiated usefully by a script.

Example code:

```
// This constructor function defines a class called Tree:
function Tree(aName, aNode1, aNode2)
{
   this.name        = aName;
   this.leftbranch  = aNode1;
   this.rightbranch = aNode2;
}

// We can now implement tree walking algorithms and
// associate them with the prototype for the tree object:
function tree_walk()
{
   if((this.leftbranch  == null) &&
      (this.rightbranch == null))
   {
      document.write(this.name);
      document.write("<BR>");
   }
   if(this.leftbranch != null)
   {
      document.write(this.name);
      document.write(" -L- ");
      this.leftbranch.walk();
   }
   if(this.rightbranch != null)
   {
      document.write(this.name);
      document.write(" -R- ");
      this.rightbranch.walk();
   }
}

// Now associate the tree walk with the prototype.
Tree.prototype.walk = tree_walk;
// Here we create a new tree object:
// In this case, we have defined null values to signify we have reached
// the end of the branching structure so we must be at a leaf node.
myTree1 = new Tree("AAA", null, null);
// Let's create another and join both to a third:
myTree2 = new Tree("BBB", null, null);
myTree3 = new Tree("CCC", myTree1, myTree2);
// Now we walk the tree
myTree3.walk();
// We could have created an array in the tree class and
// stored more than two branches.  B-Trees, Quad trees,
// and Oct trees are all useful modelling tools for
// building simulations.
```

See also:	Array simulation, Construct, `constructor` property, `prototype` property

Cross-references:

ECMA 262 edition 2 – section – 4.3.4

ECMA 262 edition 2 – section – 15.1.3

ECMA 262 edition 3 – section – 4.3.4

Wrox *Instant JavaScript* – page – 31

constructor property (Definition)

A reference to a constructor function.

Availability:	ECMAScript edition – 2

A `constructor` property is a function object that creates and initializes new objects. Each constructor has an associated prototype object that provides inheritance and shared properties.

There are `constructor` properties belonging to the `Global` object for all the Built-in (Native) object prototypes. These constructors are available as part of the core language from the `global` object. They are defined in the ECMA standard at editions 2 and 3. The host implementation may add others for you to use:

- `Object()`
- `Function()`
- `Array()`
- `String()`
- `Boolean()`
- `Number()`
- `Date()`

To create new objects, a new expression is formed with the constructor as its operand. The result is to create a new object by means of the constructor.

For any object the constructor is a property of the prototype. The prototype is a property of the object and the constructor points back at the object. In that sense the `prototype` and `constructor` properties each point at the other's parent object.

This needs to hold true for a constructor to be correctly set up:

```
myObject.prototype.constructor
```

is the same as:

```
myObject
```

They should test `true` with the `===` operator since they are supposed to be identical objects.

Warnings:

- Both Netscape and MSIE support a `constructor` property for the `Math` object. You won't find very many circumstances where you will need to create a new instance of the `Math` object. Note that the `constructor` is not a `Math` object but an `Object` object, therefore a `new Math()` statement will produce a new `Object` object and not a new `Math` object.

- In both cases, attempting to execute a `new Math()` statement will cause a run-time error. Arguably, this is a bug because objects that should not be instantiated or cloned ought not to support a constructor so that you can sensibly write general-purpose routines that can test for the existence of a `constructor` property and can then exit gracefully if it is not supported.
- The `constructor` property is supported so inconsistently across the browsers that this kind of test before use approach is almost impossible to deploy.

See also:	Construct, `Global` object, `prototype` property

Cross-references:

ECMA 262 edition 2 – section – 4.3.4

ECMA 262 edition 2 – section – 15.1.3

ECMA 262 edition 3 – section – 4.3.4

constructor.name (Definition)

The name of the constructor that created an object.

You can determine the class of an object by requesting the `name` property from its constructor. Since the constructor is a function, you are actually asking a `function` object for its name. Functions know all about their names. If you get the name of a function that's been used as a constructor, you have effectively gotten its class. Its type is still an object, but its class lets you tell one kind of object from another.

See also:	Determining the object type

Content Model (Definition)

A new model in the DOM level 3 specification describes how documents can be exchanged between implementations.

The DOM level 3 content model is embodied in the following object classes:

- `CMObject`
- `CMExternalObject`
- `CMNode`
- `CMNodeList`
- `NamedCMNodeMap`
- `CMDataType`
- `CMType`
- `ElementDeclaration`
- `ElementCMModel`
- `AttributeDeclaration`
- `EntityDeclaration`

- DocumentCM
- DomImplementationCM
- ErrorHandler
- NodeCM
- TextCM

contextual() (Method)

A JSS style control method.

Availability:	JavaScript – 3.0 Netscape – 4.0 Deprecated	
JavaScript syntax:	N	`contextual(tags.`*`anItem`*`, ...).`*`aProperty`*` = `*`aValue`*
Argument list:	*anItem*	A tag name such as `P` or `B` or `H1`
	aProperty	A style property of the returned tags object
	aValue	A value to be stored in the property of the nominated tags

This function can take a variable number of arguments, each one indicating a tag property within the tag's object. It is followed by a dot-delimited property value and basically provides a way to modify the same property across a large number of objects in a single call.

This could be a quite useful function for a variety of other non-style-related cases. You may want to experiment and see whether, in your browser, the `contextual()` method is useful. However, it may only be present in Netscape browsers, thus limiting its usefulness for deployment.

When used in a JSS context, this is more or less equivalent to `H1 P { color: red }`.

Warnings:

- This functionality is removed from Netscape 6.0.

Example code:

```
// Set the color property for several objects at once.
contextual(tags.P, tags.H1)color = "red";
```

See also:	JavaScript Style Sheets

Cross-references:

Wrox *Instant JavaScript* – page – 50

continue (Statement)

Force the next iteration of a loop.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.0 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
JavaScript syntax:	- `continue aLabelName;` - `continue;`
Argument list:	*`aLabelName`* The name of a label associated with some code

The `continue` keyword is a jump statement. It is used in an iterator loop to proceed to the next cycle without executing the remaining lines in the statement block.

A `continue` statement can only legally exist inside a `while` or `for` loop in an ECMA-compliant implementation. Implementations that provide additional iterator types may also honor the same behavior for the `continue` statement.

The `continue` statement would normally be executed conditionally, otherwise it would cause the remaining lines to be redundant since no execution flow would ever reach them. Compilers generally warn you about this, but JavaScript would likely simply ignore it.

The `continue` statement is obeyed by the smallest enclosing iterator loop.

At version 1.2 of JavaScript, the `continue` statement was enhanced to support a label as a continuing destination. When the `continue` is processed, it will jump to the start of the statement that has been labeled. If an iterator is labeled, then the `continue` is associated with that iterator. This mechanism can only be used in a `while`, `for` or `for ... in` loop.

A labeled `continue` behaves differently according to the iterator it has been used in.

Warnings:

- In Netscape 4, there is a bug with labeled `continue` statements and `do ... while` loops that causes the `continue` to vector to the top of the loop without testing the condition. This can set up an endless loop. You could work round this by creating a `while` loop and modifying the test condition.
- When the `continue` statement is used, its behavior inside a while loop suggests that a `while` loop is not exactly similar to a `for` loop. In a `while` loop, it simply runs the test condition again before deciding to loop or not. In a `for` loop, the incrementor gets executed again and then the test condition. You cannot perfectly simulate a `for` loop with a `while` loop if a `continue` statement is involved.

See also:	`break`, Completion type, `do ... while( ... )`, `for( ... ) ...`, `for( ... in ... ) ...`, Iteration statement, Jump statement, Label, `return`, Scope chain, Statement, `while( ... ) ...`

Cross-references:

ECMA 262 edition 2 – section – 10.1.4

ECMA 262 edition 2 – section – 12.7

ECMA 262 edition 3 – section – 10.1.4

ECMA 262 edition 3 – section – 12.7

Wrox *Instant JavaScript* – page – 25

Control character (Definition)

A non-printing character entity.

A control character is any character code point in the locale-specific character set that does not print a visible glyph when it is output.

See also:	ASCII, Character handling, `isCtrl()`,`isGraph()`,`isPrint()`, Printing character, Unicode

Conversion (Definition)

Changing the type of a value, object, function or constant.

Values are continuously being changed from one type of value to another inside a JavaScript interpreter. The very nature of its weak data typing and the automatic promotion and demotion of values in expressions causes implicit changes in the type of values at almost every step of a script's execution.

The conversion behavior of each type is discussed in the coverage of each of the Native primitive types that are built into the interpreter.

Refer to the `Cast` operator description as well for further information.

There are also some internal conversion operators that provide the basic underlying conversion facilities. These are described in the following topics:

- `ToBoolean`
- `ToInt32`
- `ToInteger`
- `ToNumber`
- `ToObject`
- `ToPrimitive`
- `ToString`
- `toString()`
- `ToUint16`
- `ToUint32`

These are given individual topics on account of their description in the ECMA standard. There are however four basic and fundamental conversions. These are:

- Boolean
- Number
- String
- Object

The internal function topics cover what happens inside an interpreter. In this topic and the several following, we are concerned with the script visible effects of conversion.

There are some circumstances where the conversion of an object to a number or a string can be ambiguous. The cases of concatenate or add and the relative expression operators are such an example. The `Date` objects will prefer to be converted to a string rather than a number if at all possible and consistent with the context.

Many objects have `toString()` methods. Not as many will support the `valueOf()` method. The `valueOf()` method is so named because it is not implicitly a `toNumber()` method. It may return a string because that is the most reasonable primitive. It is really a `toPrimitive()` method.

See also:	Argument, `Cast` operator, Integer constant, Integer promotion, LValue, `ToBoolean`, `ToInt32`, `ToInteger`, `ToNumber`, `ToObject`, `ToPrimitive`, `ToString`, `toString()`, `ToUint16`, `ToUint32`

Conversion to a Boolean (Definition)

Converting values to a Boolean representation.

There are three conversions to study. Obviously Boolean values will remain as they are.

This table summarizes the effects of converting values to Boolean:

Value:	Boolean equivalent:
`null`	`false`
Undefined value	`false`
Non empty string	`true`
Empty string	`false`
0	`false`
`NaN`	`false`
Infinity	`true`
Negative infinity	`true`
Any other non zero number	`true`
Object	`true`
Array	No direct boolean equivalent
Function	`true`

There is no proper Boolean equivalent for an array. The ECMA standard does not address this either. The standard does suggest that any non-null object reference convert to `true` and since an array is an object, it should rightly become a `true` value.

Warnings:

- Some non-portable behavior has been implemented in the MSIE and Netscape browsers. In some cases, the array becomes `true` in all cases. In others, a non-empty array is `true`, while an empty array is `false`. This seems to be based on the `length` value.
- You should test your target browsers to see what behavior persists if you intend to use this capability and not rely on any implicit array conversion facilities. It is probably safest to implement your own conversion method.
- The array to Boolean conversion is so inconsistently supported as to render it unusable in any cross-browser implementation.
- A more insidious side effect is exhibited by the fact that all objects convert to the Boolean `true` value, even a `Boolean` object whose value is `false`. This is demonstrated in the example. You really need to be careful if you are storing Boolean values in objects rather than simple variables, especially if they are driving some conditional code execution.

Example code:

```
<!-- Is this a bug or a feature??? -->
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myTrueObject = new Boolean(false);
if(myTrueObject)
{
   alert("True");
}
else
{
   alert("False");
}
</SCRIPT>
</BODY>
</HTML>
```

Conversion to a number (Definition)

Converting values to a numeric representation.

There are three conversions to study. Obviously numeric values will remain as they are. This table summarizes the effects of converting values to numbers:

Value:	Numeric equivalent:
`null`	`0`
Undefined value	`NaN`

Table continued on following page

Value:	Numeric equivalent:
Empty string	`0`
Numeric string	Numeric value of string
Non-numeric string	`NaN`
Boolean `true`	`1`
Boolean `false`	`0`
Object	Result of `Object.valueOf()`.
Object lacking a `valueOf()` method.	Result of conversion of result from `Object.toString()` method.
Object without `toString()` or `valueOf()` methods.	An error
Array	No direct numeric equivalent
Function	`NaN`

Converting objects to numbers will first attempt to use the `valueOf()` method and then the `toString()` method, converting the resulting string to a number after that.

There is no proper numeric equivalent for an array. The ECMA standard does not address this either.

Warnings:

- Some non-portable behavior has been implemented in the MSIE and Netscape browsers. In some cases, the array becomes `NaN`. In others, the first element is converted to a number. For some browsers this may only happen if the array is a single element long and the array is otherwise converted to `NaN`. Some browsers will take the length of the array and use that value.
- You should test your target browsers to see what behavior persists if you intend to use this capability and not rely on any implicit array conversion facilities. It is probably safest to implement your own conversion method.
- The array to number conversion is so inconsistently supported as to render it unusable in any cross browser implementation.

Conversion to a string (Definition)

Converting values to a string representation.

There are three conversions to study. Obviously string values will remain as they are.

This table summarizes the effects of converting values to strings:

Value:	Resulting string:
Zero	`"0"`
`null`	`"null"`
Undefined value	`"undefined"`
`NaN`	`"NaN"`
Infinity	`"Infinity"`
Negative infinity	`"-Infinity"`

Table continued on following page

Value:	Resulting string:
Numeric value	That numeric value as a sequence of characters.
Boolean `true`	`"true"`
Boolean `false`	`"false"`
Object	Result of `Object.toString()`.
Object lacking a `toString()` method.	Result of conversion of result from `Object.valueOf()` method.
Object without `toString()` or `valueOf()` methods.	An error
Array	Comma elements joined by a comma.
Function	Depends on implementation.

Converting objects to strings will first attempt to use the `toString()` method and then the `valueOf()` method, converting the resulting number to a string after that.

Conversion to an object (Definition)

Converting values to a object-structured representation.

There are three conversions to study. Obviously Object values will remain as they are.

This table summarizes the effects of converting values to objects:

Value:	Object representation:
`null`	An error
Undefined value	An error
Empty string	`String` object
Non-empty string	`String` object
`0`	`Number` object
`NaN`	`Number` object
Infinity	`Number` object
Negative infinity	`Number` object
Any other non-zero number	`Number` object
Boolean `true`	`Boolean` object
Boolean `false`	`Boolean` object

Cookie (Advice)

A means of maintaining state in the client machine by storing information in a cookie that is associated with the document.

A cookie is a small fragment of textual data that is associated with the current page. It is not modeled particularly well for access by JavaScript and you will need to use some scripting to disassemble and reassemble the name-value pairs that are concatenated together to make the cookie.

The `cookie` property for a `document` returns a string containing ALL the cookies that apply to the document. You will need to split them into individual cookies by separating them at semicolon boundaries. From there you will need to obtain for each cookie a `name=value` construct that you can further dismantle and process.

Note that you will not get any of the special attributes of the cookie since they are write only. The only thing you can get back is its `value` property.

When you read the `cookie` property from a document, you get a list of cookie values, one for each cookie. When you assign to the `cookie` property of a document, you define only a single cookie whose name is the first name-value pair in the string. Other attributes are appended by concatenation and delimited from one another by a semicolon. The other attributes are optional and all that is required is the new cookie name and value pair.

You must make sure the cookie value you assign is properly URL escaped so that it can be sent back to the server when necessary.

Cookie values should be defined as a series of `name=value` pairs. As well as the fundamental cookie value, you can define the following `name=value` pairs:

- `expires`
- `path`
- `domain`
- `secure`

Cookies can be redefined whenever you want. To delete a cookie, set it to some value with an expiry date defined to be prior to the current date and time.

Cookies are useful but you should not rely on them 100%. It is possible that the user may have disabled cookie support in their browser. You should also avoid storing sensitive information in a cookie and it is intended that the value you store there is small. Storing huge amounts of data in a cookie is somewhat bad etiquette.

The example code demonstrates how two functions (`setCookie` and `getCookie`) can be created to simplify the creation and parsing of cookies. The comments in the example explain how they work.

Warnings:

- There are limitations to the size and quantity of cookies that can be used. Typically these days browsers will exceed these limitations, but you should always code for the minimum specification.
- The specification for cookies says that browsers need support no more than 300 different cookies. Of those, no more than 20 should be associated with any particular server. For each cookie, the data stored is not expected to exceed 4 kilobytes.
- You should therefore try to store as much data in a single cookie as you can for a given web site and not create different cookies for each page. Indeed for a properly designed data-driven site, it is hard to see how you would need more than one single cooking containing a short but unique user identifier. As long as you can look up that user's record quickly when a new session commences, all session and user state information can then be maintained on the server unless you want to traverse through some static pages. Even then state can be carried by means of cookies and special web-server code to intercept the request-response loop.

Example code:

```
// Example provided by Martin Honnen
// A function that sets cookie values properly
// The cookieName and cookieValue arguments are mandatory
// but all other arguments are optional.
// The expires argument is a Date object.
// The path defines the part of the document tree on the server
// that the cookie is valid for.
// The domain argument allows multiple server hosts to be used.
// The secure value is boolean and only applicable for use
// with HTTPS: connections.
function setCookie(cookieName, cookieValue, expires, path, domain, secure)
{
   document.cookie = escape(cookieName) + '=' + escape(cookieValue)
      + (expires ? '; EXPIRES=' + expires.toGMTString() : '')
      + (path ? '; PATH=' + path : '')
      + (domain ? '; DOMAIN=' + domain : '')
      + (secure ? '; SECURE' : '');
}

// A complementary function to unwrap a cookie.
function getCookie(cookieName)
{
   var cookieValue = null;
   var posName = document.cookie.indexOf(escape(cookieName) + '=');

   if (posName != -1)
   {
      var posValue = posName + (escape(cookieName) + '=').length;
      var endPos = document.cookie.indexOf(';', posValue);
      if (endPos != -1)
      {
         cookieValue = unescape(document.cookie.substring(posValue, endPos));
      }
      else
      {
         cookieValue = unescape(document.cookie.substring(posValue));
      }
   }
   return cookieValue;
}

// Tryout 1: Set a session cookie which expires after
//           the browser is closed
setCookie ('TRYOUT', '1');

// Tryout 2: Set a cookie which expires after 24 hours
var now = new Date();
var tomorrow = new Date(now.getTime() + 1000 * 60 * 60 * 24);
setCookie ('TRYOUT', '2', tomorrow);

// Tryout 3: Set a cookie with a path
setCookie ('TRYOUT', '3', null, '/');

// Tryout 4: Delete a cookie by setting its expiry date to
//           be sometime in the past
var now = new Date();
var yesterday = new Date(now.getTime() - 1000 * 60 * 60 * 24);
setCookie('TRYOUT', '4', yesterday);
```

See also:	`Document.cookie`

Web-references:

http://www.netscape.com/newsref/std/cookie_spec.html

Cookie domain (Attribute)

An attribute that defines the domain scope of a cookie.

For security reasons, cookies can only be sent back to the web-server the creating document originated from. However, in large web-server farms, the web pages may be distributed for load balancing reasons. Because of this, the domain attribute provides a way to widen the scope to any machine within a domain.

Cookie expires (Attribute)

An attribute that defines the expiry date and time of a cookie.

The lifetime of a cookie is defined with this attribute. From JavaScript, you can set this attribute but you cannot read it since there is no real object model for the cookies.

To define this attribute, you add a name-value pair to the cookie definition string whose name is "expires" and whose value is defined according to the `Date.toGMTString()` method.

A cookie only survives for the duration of the page in the browser unless you define an expiry date for it. If the expiry date is in the future when the browser exits, they will be remembered in a persistent cache inside the browser. If not, the cookie is discarded and unavailable next time you run the browser.

See also:	`Date.toGMTString()`

Cookie path (Attribute)

An attribute that defines the path scope of a cookie.

The scope of a cookie can be limited to a certain part of the document tree within the web server. By defining a node within the document hierarchy, the cookie will only be sent to the web server when requesting a page that exists at that path or lower down in any sub-directories within it.

Unless you specify this value, the cookie will by default be available to any pages in the same directory as the page that created it or in pages lower down in sub-directories. These might be referred to as sibling or child pages. The path value is usually modified to be more inclusive than the default settings.

Cookie secure (Attribute)

A Boolean attribute that defines whether a cookie is secure or not.

The secure attribute is a Boolean value that defines whether the secure protocol is required. If it is activated, then the cookie is only sent to a server when the secure `https:` protocol is used. To activate this facility simply add the secure attribute to the cookie.

Cookie value (Attribute)

An attribute containing the value of a cookie.

Refer to:

Cookie

Copying objects (Advice)

Object references are normally duplicated in preference to the objects themselves.

When you assign the result of a new operation to a variable, you are actually storing a reference to the object. Copying the contents of one variable to another by assignment copies the reference and not the object.

Creating a new object and copying all the top-level properties across from the original is a shallow copy, so called because it only goes one layer into the hierarchy.

If you are able to create a new instance of the object and then recursively copy all the properties across from one to the other, you will have made a deep copy of the object. To do this, you need to traverse all of the branches of the object references, instantiating new copies of any objects you find. This may or may not be possible since some properties may be hidden from view, may be read-only or may not be enumerable and therefore impossible to copy without knowing what they are.

In practice though, any properties you put into the original object from a script are likely to be copyable and you are likely to know enough about your objects to be able to accomplish this at least to the extent that you need.

Comparing two objects may yield a different result depending on whether you are comparing shallow copies or deep copies. Comparing two shallow copies is actually the act of comparing references. If they both refer to the same object, then they must be identical and equal. Comparing deep copies requires a comparison on a property-by-property basis. This may prove that the objects are equal but not identical. A simple compare of the references would prove `false`.

To compare deep copies, you should implement an object method called `isEqualTo()` and pass the object that you want to compare. The receiving object can then enumerate its properties and test the passed object for the existence and content of those properties.

You may need to make your deep copy algorithm recursive if you are copying objects that contain references to objects. For example, `Arrays` of `Arrays`.

Writing a generalized algorithm is quite difficult if you want it to work across browsers. This is because you need to be able to determine the class of any objects you encounter so that you can perform the right kind of copying on them. This might be easier if you limit the kind of objects you use.

Example code:

```
// Copying only an object reference
myObject1 = new Array("AAA", "BBB", "CCC");
myObject2 = myObject1;

// Shallow copying (one layer deep)
myObject1 = new Array("AAA", "BBB", "CCC");
```

```
myObject2 = new Array(myObject1.length);
for(ii=0; ii < myObject1.length; ii++)
{
   myObject2[ii] = myObject1[ii];
}

// Deep copying (knowledge of the internal structure required)
myItem1   = new Array("A", "B", "C");
myItem2   = new Array("1", "2", "3", myItem1);
myObject1 = new Array("X", "Y", "Z", myItem2);
myObject2 = new Array();
myObject2[0] = myObject1[0];
myObject2[1] = myObject1[1];
myObject2[2] = myObject1[2];
myObject2[3] = new Array();
myObject2[3][0] = myObject1[3][0];
myObject2[3][1] = myObject1[3][1];
myObject2[3][2] = myObject1[3][2];
myObject2[3][3] = new Array();
myObject2[3][3][0] = myObject1[3][3][0];
myObject2[3][3][1] = myObject1[3][3][1];
myObject2[3][3][2] = myObject1[3][3][2];
```

See also:	Multi-dimensional arrays

Core JavaScript (Definition)

That part of the language that is deemed to be fundamental.

The core part of the language includes the operators and basic language elements. This covers constructs such as `for` loops, `while` loops, `if/else` conditions and `switch/case` trees.

The `Global` object is included in the core language, but gets extended with additional functionality when the interpreter is hosted.

The prototype-inheritance mechanisms, scope chain, and function call support is part of the core language too, as are the constructor frameworks.

These are built around the generic set of objects that represent the primitive data types such as Number, Boolean, String, and Array.

The core language also includes the `Math` and `Date` object support.

Core Object (Definition)

Objects that are built into the base language.

Core objects describe the fundamental primitive data types such as `Number`, `Boolean`, `String`, and `Array`. The `Object` object, which is considered to be the super-class of all objects, is a fundamental core object.

`Math` and `Date` are core objects too and provide support for mathematical capabilities and date handling respectively.

Cross-browser compatibility (Definition)

Different browsers have different features and capabilities.

The differences between browsers are most acute when considering Netscape vs. MSIE. The Opera and iCab browsers are keen to become ECMAScript-compliant. Recent versions of MSIE also make this claim and it is likely that imminent versions of Netscape will be at least as ECMAScript-compliant as the other browsers. This bodes well for the future in that we can code for a base level of functionality and be able to deploy a useful sub-set of the JavaScript capabilities of each browser.

Inevitably, they will continue to present features that are browser-specific. Some of those will be accommodated in the next revision of the ECMA standard. But some won't.

You may find that your script needs to do something more esoteric than a simple form validation or mouse rollover. In particular doing any dynamic HTML and changing `<DIV>`, `<SPAN>`, or `<LAYER>` content. For these kinds of things, you may need to use browser-detection scripts and then provide alternative versions of some of your functions so that you select one that is appropriate for the browser the script happens to find itself running in.

See also:	Compatibility, Defensive coding

Cross-platform compatibility (Definition)

Portability is a major issue for web developers.

When you write JavaScript that exploits capabilities of either Netscape or MSIE, you may already be taking account of the differences between the two browsers.

However, those browsers, even at the same versions, may not always behave identically on different platforms.

Netscape is available on more platforms than MSIE. Opera and iCab are likely to be available as widely as Netscape, but both are stressing their adherence to the standards and both make bold claims to be compliant. That suggests that their behavior across computing platforms will be identical.

In actual fact, there are minor differences even on the best crafted browser projects. MSIE and Netscape fall some way behind the lesser known browsers in this respect.

If you are deploying a major web site that contains JavaScript, you really must consider setting up a test suite to check the cross-platform capabilities.

It is possible to install a variety of browsers onto a single Macintosh system. It is a little more difficult with Windows installations, which generally do not cope well with more than one version of any particular product being installed. You experience difficulties installing older versions of MSIE onto recent versions of Windows that already have an embedded web browser.

You could, if you are serious about cross platform testing, easily end up with a test suite of 10 or more machines. That way you would likely be able to cover Linux as well.

Although you may install many browser applications on a single Macintosh, you may want to add more hard disks and install multiple operating system versions so you can multiple-boot the machine.

This is because some cross-platform differences are due to underlying operating system level code, which may be revised with a new release of the OS and which may manifest itself as an artifact on the web page when it is displayed.

The `navigator` property tells you a lot about the browser and platform. However, MSIE supports other platform and browser detection methods that are not available on other browsers.

See also:	Compatibility, `Navigator` object

crypto (Property)

A reference to a `Crypto` object for security encoding.

Availability:	JavaScript – 1.2 Netscape – 4.04	
Property/method value type:	`Crypto` object	
JavaScript syntax:	N	`crypto`
	N	`myWindow.crypto`

Property attributes:

`ReadOnly`.

Refer to:

`Window.crypto`

Crypto object (Object/Navigator)

An object to manage cryptographic resources.

Availability:	JavaScript – 1.2 Netscape – 4.04	
JavaScript syntax:	N	`crypto`
Object properties:	`constructor`	
Object methods:	`random(),signText()`	

This object appears to be quite opaque. That seems to make sense as you wouldn't want security-related information to be visible simply by enumerating the properties.

There appear to be no enumerable properties belonging to this object.

See also:	`Window.crypto`

Property	JavaScript	JScript	N	IE	Opera	Notes
constructor	1.2 +	-	4.04 +	-	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
random()	1.2 +	-	4.04 +	-	-	-
signText()	1.2 +	-	4.04 +	-	-	-

Crypto.constructor (Property)

A constructor function for the `Crypto` object in Netscape.

Availability:	JavaScript – 1.2 Netscape – 4.04	
Property/method value type:	`Crypto` object	
JavaScript syntax:	N	`crypto.constructor`

You can use this as a way of creating `Crypto` objects.

This property is useful if you have an object that you want to clone, but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

Crypto.random() (Function)

Return a string of randomly generated characters.

Availability:	JavaScript – 1.2 Netscape – 4.04	
JavaScript syntax:	N	`crypto.random()`

Refer to:

Security policy

Crypto.signText() (Function)

Request a digital signature from a user.

Availability:	JavaScript – 1.2 Netscape – 4.04	
JavaScript syntax:	N	`crypto.signText()`

Refer to:

Security policy

Cryptoki (Security related)

Part of the Netscape security facilities.

Cryptoki is pronounced crypto-key and is short for cryptographic token interface. It is provided by RSA Data Security, Inc and can be accessed from the C language and Java.

See also:	`Pkcs11` object

CSS (Standard)

Cascading Style Sheets.

In the early days of the web, page designers were intent on creating ever more complex HTML in an effort to be able to organize how the content would appear on every browser. It became important to be able to render the page with pixel perfect accuracy. This was not the original intent of HTML.

With the introduction of CSS, designers were given a much larger range of tools with which to control the appearance of web pages. For some time CSS level 1 was thought to be sufficient. Eventually this was superseded by CSS level 2.

The Netscape 4 browser supports a third alternative, that of JavaScript Style Sheets or JSS for short. With the release of Netscape 6.0 providing standards-based support, JSS has no future and should not be used in any new projects. Our JSS coverage has accordingly been marked as deprecated.

CSS is constructed from packages of rules, which are assembled into style-sheets.

There is no object model defined for style sheets as of DOM level 1. Until this is ratified in a later DOM specification, Netscape and MSIE continue to support mutually incompatible style sheet API specifications.

The DOM level 2 implementation of CSS style objects (the CSS Object model) provides a complex hierarchy of objects. These are only partly implemented in current browsers. The most complete implementation is in the MSIE browser, and even then the objects are factored differently and the classes named in an MSIE-specific manner.

The DOM CSS suite is embodied in the following classes:

- `CSSStyleSheet`
- `CSSRuleList`
- `CSSRule`
- `CSSStyleRule`
- `CSSMediaRule`
- `CSSFontFaceRule`
- `CSSPageRule`
- `CSSImportRule`

- `CSSCharsetRule`
- `CSSUnknownRule`
- `CSSStyleDeclaration`
- `CSSValue`
- `CSSPrimitiveValue`
- `CSSValueList`
- `RGBColor`
- `Rect`
- `Counter`
- `ViewCSS`
- `DocumentCSS`
- `DOMImplementationCSS`
- `ElementCSSInlineStyle`
- `CSS2Properties`

See also:	`<STYLE>`, CSS level 1, CSS level 2, CSS-P, Dynamic HTML

CSS level 1 (Standard)

A standard for describing style sheets.

The CSS level 1 standard was issued in December 1996 and describes a simple formatting model intended mainly for screen, based presentations. It makes available approximately 50 properties for controlling the appearance of a web page.

See also:	`<STYLE>`, CSS, CSS level 2, JavaScript Style Sheets

Web-references:

http://www.w3.org/TR/REC-CSS1

CSS level 2 (Standard)

A standard for describing style sheets.

The CSS level 2 standard was presented around May 1998 and was based on the earlier CSS1 standard. It adds another 70 properties to the 50 already available with CSS level 1.

See also:	`<STYLE>`, CSS, CSS level 1, CSS-P, Dynamic positioning

Web-references:

http://www.w3.org/TR/REC-CSS2

CSS-P (Standard)

Specifically and only the positional controls for HTML entities, nowadays folded into the CSS level 2 standard.

Warnings:

- The CSS-P specification allows that any tag can be positioned. However Netscape 4 only supports the positioning of elements that have an opening and closing tag. This means you cannot control the position of an `IMG` object on its own unless you encapsulate it correctly within the document. You can control the posistion of an `IMG` object if it is enclosed within a set of balanced `<SPAN>`, `<DIV>`, or `<A>` HTML tags.
- Netscape 4 also converts any absolutely positioned `<DIV>` tags into layers. This means you can manipulate them as layers but this requires special script code that is only usable on version 4 of Netscape. This has completely changed on Netscape 6.0.

See also:	CSS, CSS level 2, Dynamic positioning

Currency symbol (Definition)

A symbol that denotes a locale-specific currency.

Many (but not all) currency symbols are defined as Unicode character code points. The currently extant standard does not adequately support the Euro symbol in common use, although it is generally available in the fonts that ship with MacOS and Windows.

Here is a list of the Unicode defined currency symbols at this time:

Code:	HTML:	Description:
0023	&#035;	Hash sign
0024	&#036;	USA (Dollar)
00A2	&#162;	USA (Cent)
00A3	&#163;	United Kingdom (Pound)
00A4	&#164;	General currency symbol
00A5	&#165;	Japan (Yen)
0192	&#131;	Florin
0E3F	-	Thailand (Baht)
20A0	-	Euro currency sign
20A1	-	Costa Rica & El Salvador (Colon)
20A2	-	Brazil (Cruzeiro)
20A3	-	France (Franc)
20A4	£	Italy (Lira - similar to 00A3)
20A5	-	USA (Mill - 1/10 cent)

Table continued on following page

Code:	HTML:	Description:
20A6	-	Nigeria (Naira)
20A7	-	Spain (Peseta)
20A8	-	India (Rupee - can be drawn as Rs)
20A9	-	Korea (Won)
20AA	-	Israel (New Sheqel)
20AB	-	Vietnam (Dong)

See also:	Localization

currentStyle object (Object/JScript)

An object that represents the cascaded format and style of its parent object.

Availability:	JScript – 5.0 Internet Explorer – 5.0
Inherits from:	style object
JavaScript syntax:	IE *myCurrentStyle* = *myElement*.currentStyle
Object methods:	getAttribute(),getExpression(),removeExpression(), setAttribute(),setExpression()

Because the style values are cascaded from style sheet to style sheet and may include some inline styles as well as some explicit styles, objects need to maintain a current style value that is the result of all the inheritances applied on top of one another.

In addition they maintain a runtime style that reflects dynamic changes as well. The runtime style is based on the current style in the first place.

This represents the cascaded format and style of its parent object.

The properties belonging to this object correspond closely to those of the style object and so there is little point in discussing them again here. Refer to the style object property descriptions for details of the various properties.

See also:	Element.currentStyle, Element.runtimeStyle, runtimeStyle object, style object (2)

Method	JavaScript	JScript	N	IE	Opera	Notes
getAttribute()	-	5.0 +	-	5.0 +	-	-
getExpression()	-	5.0 +	-	5.0 +	-	-
removeExpression()	-	5.0 +	-	5.0 +	-	-
setAttribute()	-	5.0 +	-	5.0 +	-	-
setExpression()	-	5.0 +	-	5.0 +	-	-

Cursor object (Object/NES)

This object encapsulates a cursor that was returned from the database as a result of an SQL query.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	*myCursor* = Connection.cursor(*aQuery*)
	NES	*myCursor* = Database.cursor(*aQuery*)
Argument list:	*aQuery*	A valid SQL query for the database
Object properties:	<column_name>	
Object methods:	blobImage(),blobLink(),close(),columnName(),columns(), deleteRow(),insertRow(),next(),updateRow()	

You can construct a `cursor` object by requesting it via the `Connection` object or `database` object, both of which have `cursor` methods.

Example code:

```
<SERVER>
// An example cursor retrieved from a database
// Based on the example from Wrox Professional JavaScript
database.connect("ODBC", "myDatabase", "", "", "");
myCursor = database.cursor("SELECT * FROM MY_TABLE");
</SERVER>
```

See also:	Connection.cursor(),database object, database.cursor(), Netscape Enterprise Server, unwatch(),watch()

Property	JavaScript	JScript	NES	Notes
<column_name>	1.1 +	-	2.0 +	-

Method	JavaScript	JScript	NES	Notes
blobImage()	1.1 +	-	2.0 +	-
blobLink()	1.1 +	-	2.0 +	-
close()	1.1 +	-	2.0 +	-
columnName()	1.1 +	-	2.0 +	-
columns()	1.1 +	-	2.0 +	-
deleteRow()	1.1 +	-	2.0 +	-
insertRow()	1.1 +	-	2.0 +	-
next()	1.1 +	-	2.0 +	-
updateRow()	1.1 +	-	2.0 +	-

Cursor.<column_name> (Property)

The columns within the answer set are reflected into properties with the same names.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*myCursor.aColumnName*
Argument list:	*aColumnName*	The name of a column within the answer set

The column names are inherited from the tables in the database that the cursor is traversing.

Cursor.blobImage() (Method)

This method creates an `<IMG>` element having the appropriate mimeType for the `blob` object.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Image` object	
JavaScript syntax:	NES	*myCursor*.blobImage(*aFormat*)
	NES	*myCursor*.blobImage(*aFormat*, *aTxt*)
	NES	*myCursor*.blobImage(*aFormat*, *aTxt*, *anAlign*)
	NES	*myCursor*.blobImage(*aFormat*, *aTxt*, *anAlign*, *aPixWid*)
	NES	*myCursor*.blobImage(*aFormat*, *aTxt*, *anAlign*, *aPixWid*, *aPixHgt*)
	NES	*myCursor*.blobImage(*aFormat*, *aTxt*, *anAlign*, *aPixWid*, *aPixHgt*, *aPixBrdr*)
	NES	*myCursor*.blobImage(*aFormat*, *aTxt*, *anAlign*, *aPixWid*, *aPixHgt*, *aPixBrdr*, *isMap*)
Argument list:	*aFormat*	Image file format
	anAlign	The alignment of the image
	aPixBrdr	The border value
	aPixHgt	The height of the image
	aPixWid	The width of the image
	aTxt	The alt text for the image
	isMap	Whether the image is a map

The data is pulled out of the database according to the specified parameters. The BLOB can then be displayed as if it were an image in an `<IMG>` tag.

The format argument should contain an image specifier such as "GIF" or "JPEG" that can map conveniently to a file extension or MIME type.

In general, the remaining parameters to this method correspond to the HTML tag attributes that can be used with an <IMG> tag and are optional.

See also:	blob.blobImage()

Cursor.blobLink() (Method)

This method creates an <A> element that links to the BLOB data.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	Anchor object	
JavaScript syntax:	NES	*myCursor*.blobLink(*aMimeType*, *aString*)
Argument list:	*aString*	The text inside the link
	aMimeType	The MIME type of the document being displayed

The data is pulled out of the database according to the specified parameters. The BLOB can then be displayed as if it were an document in an <A> tag.

See also:	blob.blobLink()

Cursor.close() (Method)

To close the cursor.

Availability:	JavaScript - 1.1 Netscape Enterprise Server - 2.0	
JavaScript syntax:	NES	*myCursor*.close()

After a cursor has been closed, there can be no further access to it until it is reopened by creating a new instance from the cursor property belonging to the Connection or database objects.

All open cursors are automatically closed by NES at the end of the client request.

Cursor.columnName() (Method)

Returns the name of the indexed column within the cursor.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	String primitive	
JavaScript syntax:	NES	*myCursor*.columnName(*anIndex*)
Argument list:	*anIndex*	A valid column number within the cursor

The columns are arranged in order as they appear in the answer set. This method indexes the name list in the answer set and returns the names for each column using a zero-based index.

Cursor.columns() (Method)

Returns the number of columns in the cursor.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*myCursor*`.columns()`

The answer set contains a list of columns that were retrieved from the database. This method returns a count of the columns in the answer set.

Cursor.deleteRow() (Method)

Deletes the current row in the cursor.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*myCursor*`.deleteRow(`*aTableName*`)`
Argument list:	*aTableName*	The name of a table to have a record deleted from it

If the cursor is updateable, then you can remove a row from the table. The row removed corresponds to the one the cursor is currently referencing.

The row following the current row becomes the current row unless the cursor is on the last row of the cursor in which case it is indexed backwards to the previous row.

Refer to the Status Code topic for a list of the status code values that are returned by this method.

See also:	Status Code

Cursor.insertRow() (Method)

A new row is inserted into the table that the cursor is pointing at.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*myCursor*`.insertRow(`*aTableName*`)`
Argument list:	*aTableName*	The name of a table to have the new data inserted into it

New rows can be inserted into the table if the cursor is updateable.

Note that there is no guarantee as to the position of the new inserted record. There may be automatic sorting triggers activated in the database that cause it to be sorted into the correct position or the new record may simply be appended to the end. It is implementation-dependent.

Refer to the Status Code topic for a list of the status code values that are returned by this method.

See also:	Status Code

Cursor.next() (Method)

Index the cursor onwards to the next row.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	*myCursor*.next()

The cursor is moved onwards to point at the next sequential record.

When the cursor reaches the end of the selection, it cannot be indexed any further onwards. What happens in this case is implementation-dependent, but there should probably be an error generated.

Cursor.prototype (Property)

The prototype for the `Cursor` object that can be used to extend the interface for all `Cursor` objects.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Cursor` object	
JavaScript syntax:	NES	`Cursor.prototype`
	NES	*myCursor*`.constructor.prototype`

Refer to:

`prototype` property

Cursor.updateRow() (Method)

Any pending changes to the current row are saved back to the database.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0
Property/method value type:	`Number primitive`

JavaScript syntax:	NES	*myCursor*.updateRow(*aTableName*)
Argument list:	*aTableName*	The name of a table to be updated with the new data

If the cursor is updateable, then the row currently referred to by the cursor can be updated.

You may need to perform a commit action on the database to confirm the changes and avoid them being rolled back when the session is closed. Whether this is necessary depends on whether there are auto-confirm triggers or whether the target database even supports a two phase commit.

Refer to the Status Code topic for a list of the status code values that are returned by this method.

See also:	Status Code

Custom object (Definition)

A user-defined object inheriting the properties of an `Element` object within the MSIE browser model.

Refer to:

`Element` object

Data Type (Definition)

The type of data contained in a variable or described by a literal.

Fundamentally, JavaScript works with the following value types:

- Numbers
- Strings
- Boolean values
- Objects

All other data types are aggregates of those.

Objects are really collections of primitive values and are accessed by reference.

See also:	Boolean, Number, Object, String

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page –34

Data-tainting (Security related)

A mechanism for marking data in the client and controlling its use. An obsolete security work-around.

The data-tainting model was implemented in Netscape 3 but deprecated by version 4. It was never implemented in MSIE.

Rather than prevent access to data in other parts of the browser space, it allows full access even to private data. However, that access marked the data as tainted and any values that were derived from it were also tainted. Tainted data values could not be sent back to the server and in fact were not permitted to leave the client.

These capabilities were not used very much in production systems and have now been superseded by the signed scripts and privilege model.

Warnings:

- This is deprecated and should not be used in new projects.

See also:	Restricted access, Security policy, Signed scripts

database object (Object/NES)

An object that encapsulates the access to a back end database from Netscape Enterprise Server.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0
JavaScript syntax:	NES `database`
Object properties:	`prototype`
Object methods:	`beginTransaction()`, `commitTransaction()`, `connect()`, `connected()`, `cursor()`, `disconnect()`, `execute()`, `majorErrorCode()`, `majorErrorMessage()`, `minorErrorCode()`, `minorErrorMessage()`, `rollbackTransaction()`, `SQLTable()`, `storedProc()`, `storedProcArgs()`, `toString()`

The `database` object is always available through the `database` property of the `Global` object within the Netscape Enterprise Server session. Before using any other methods belonging to this object, you must first successfully connect to a database.

Until the `database` object has made a connection to the database, the only methods that have any meaning are `database.connect()` and `database.connected()`.

See also:	`Cursor` object, Netscape Enterprise Server, `response.database`, `unwatch()`, `watch()`

Property	JavaScript	JScript	NES	Notes
`prototype`	1.1 +	-	2.0 +	-

Method	JavaScript	JScript	NES	Notes
`beginTransaction()`	1.1 +	-	2.0 +	-
`commitTransaction()`	1.1 +	-	2.0 +	-
`connect()`	1.1 +	-	2.0 +	-
`connected()`	1.1 +	-	2.0 +	-
`cursor()`	1.1 +	-	2.0 +	-
`disconnect()`	1.1 +	-	2.0 +	-
`execute()`	1.1 +	-	2.0 +	Warning
`majorErrorCode()`	1.1 +	-	2.0 +	-
`majorErrorMessage()`	1.1 +	-	2.0 +	-
`minorErrorCode()`	1.1 +	-	2.0 +	-

Method	JavaScript	JScript	NES	Notes
`MinorErrorMessage()`	1.1 +	-	2.0 +	-
`rollbackTransaction()`	1.1 +	-	2.0 +	-
`SQLTable()`	1.1 +	-	2.0 +	-
`storedProc()`	1.1 +	-	3.0 +	-
`storedProcArgs()`	1.1 +	-	3.0 +	-
`toString()`	1.1 +	-	2.0 +	-

database.beginTransaction() (Method)

Marks the beginning of a transaction with the database.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	`database.beginTransaction()`

This commences a transaction on the database. The transaction methods are useful if you want to maintain control over the two phase commit process in the database. The alternative `database.execute()` method can accomplish the same but it will automatically commit any changes as the SQL is executed. This transaction based technique provides a finer degree of control and allows for rollbacks.

Refer to the Status Code topic for a list of the status code values that are returned by this method.

See also:	`database.commitTransaction()`, `database.rollbackTransaction()`, Status code

database.commitTransaction() (Method)

Commit the changes made during this transaction.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	`database.commitTransaction()`

In the transaction based mechanism, you can control the commit and rollback of the transaction if necessary. You will need to explicitly call this method to commit the changes to the database.

Refer to the Status Code topic for a list of the status code values that are returned by this method.

See also:	`database.beginTransaction()`, `database.rollbackTransaction()`, Status code

database.connect() (Method)

Forms a connection to the database using the database type to select the correct one.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	`database.connect(aType, aServer, aUser, aPassword, aDb);`
	NES	`database.connect(aType, aServer, aUser, aPassword, aDb, maxCon);`
	NES	`database.connect(aType, aServer, aUser, aPassword, aDb, maxCon, aFlag);`
Argument list:	*aDb*	The name of a database
	aFlag	Commit or roll back on close
	aPassword	A valid password for the user
	aServer	The name of a database server
	aType	A valid connection type
	aUser	A user registered for database access on the server
	maxCon	The maximum number of simultaneous connections

When connecting to a database, you need to indicate the type of database you are connecting to. The following are examples of commonly available database types:

- ORACLE
- SYBASE
- INFORMIX
- DB2
- ODBC

Use one of these values in the first argument to this method.

You will also need to know the name of your target server, a valid username and password and if multiple databases are supported by your database server, then you will need to know the name of the target database you want to connect to.

The last two arguments indicate the maximum number of connections available at once and a flag to indicate the commit policy on closure. You can elect to automatically commit any changes (dangerous) or roll back any uncommitted changes.

See also:	`DbPool` object, `DbPool()`, `DbPool.connect()`

database.connected() (Method)

A flag, indicating the connection status for this `database` object.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	`database.connected()`

This method returns a Boolean value that tells you whether the database object is connected to a database or not.

See also:	`DbPool.connected()`

database.cursor() (Method)

Creates a new cursor object with the SQL supplied in the argument.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Cursor object`	
JavaScript syntax:	NES	`database.cursor(`*`aQuery`*`)`
	NES	`database.cursor(`*`aQuery, aFlag`*`)`
Argument list:	*`aFlag`*	Defines whether the cursor is updateable or not
	`aQuery`	A valid SQL query for the database

This method returns an answer set as a `cursor` object.

The updateable flag controls whether the cursor can be updated or not. Setting it `true` means the cursor can be used to send an update change to the database. `False` renders it read only.

See also:	`Cursor` object

database.disconnect() (Method)

Severs the current connection to the database.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
JavaScript syntax:	NES	`database.disconnect()`

It is a good idea to disconnect from the database when you know you won't need it anymore. This is good practice and allows other processes to connect when resources are scarce.

Until the database is connected again, only the `connect()` and `connected()` methods have any meaning.

See also:	`DbPool.disconnect()`

database.execute() (Method)

Executes the SQL passed in the argument.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	`database.execute(`*`someSQL`*`)`
Argument list:	*`someSQL`*	A fragment of valid SQL for the database

This method cannot return an answer set. It does return a status code indicating how successfully it managed the query.

Refer to the Status Code topic for a list of the status code values that are returned by this method.

Warnings:

- This method will automatically perform a commit on any SQL that you send. You can avoid this by using transaction methods.

See also:	Status code

database.majorErrorCode() (Method)

Returns the error code for an error that may have happened in the database or the interface to it.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`database.majorErrorCode()`

For a status code value of 5 when using the Oracle database, this yields a return code from the Oracle Call-level interface.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the SQL server message number.

For a status code value of 7 when using the Informix database, this yields the Informix error code.

For a status code value of 7 when using the Sybase database, this yields the DB-Library error number.

See also:	`Connection.majorErrorCode()`, `DbPool.majorErrorCode()`, Error handling, Status code

database.majorErrorMessage() (Method)

Returns the error message text for an error that may have happened in the database or the interface to it.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`database.majorErrorMessage()`

For a status code value of 5 when using the Oracle database, this yields a text string describing the server error.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields a text string from SQL server.

For a status code value of 7 when using the Informix database, this yields the text string from the vendor error library.

For a status code value of 7 when using the Sybase database, this yields a text string from the DB-Library.

See also:	`Connection.majorErrorMessage()`, `DbPool.majorErrorMessage()`, Error handling, Status code

database.minorErrorCode() (Method)

Returns a supplementary error code for an error that may have happened in the database or the interface to it.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`database.minorErrorCode()`

For a status code value of 5 when using the Oracle database, this yields an operating system error code from the Oracle Call-level interface.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the severity level from SQL server.

For a status code value of 7 when using the Informix database, this yields the ISAM error code.

For a status code value of 7 when using the Sybase database, this yields the severity level of the error from the DB-Library.

See also:	`Connection.minorErrorCode()`, `DbPool.minorErrorCode()`, Error handling, Status code

database.minorErrorMessage() (Method)

Returns a supplementary error message text for an error that may have happened in the database or the interface to it.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`database.minorErrorMessage()`

For a status code value of 5 when using the Oracle database, this yields the Oracle server name.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the SQL server name.

For a status code value of 7 when using the Informix database, this yields a text string describing the ISAM error.

For a status code value of 7 when using the Sybase database, this yields the text of the operating system error from the DB-Library.

See also:	`Connection.minorErrorMessage()`, `DbPool.minorErrorMessage()`, Error handling, Status code

database.prototype (Property)

The prototype for the database object that can be used to extend the interface for all database objects.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`database object`	
JavaScript syntax:	NES	`database.prototype`

See also:	`Prototype` property

database.rollbackTransaction() (Method)

Undoes any changes made in the current transaction.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	`database.rollbackTransaction()`

In the transaction based technique, this method provides a way to rollback the changes if you have not yet committed them.

Refer to the Status Code topic for a list of the status code values that are returned by this method.

See also:	`database.beginTransaction()`, `database.commitTransaction()`, Status code

database.SQLTable() (Method)

Creates an HTML table based on the results of the SQL query provided in the argument.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`String` primitive	
JavaScript syntax:	NES	`database.SQLTable(`*`someSQL`*`)`
Argument list:	*`someSQL`*	A valid SQL query for the connected database

The select statement is passed to the database and the results are returned to the caller as a series of records formatted to appear as an HTML table.

database.storedProc() (Method)

Creates a stored procedure object and runs the specified stored procedure in the database.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 3.0	
Property/method value type:	`Stproc` object	
JavaScript syntax:	NES	`database.storedProc(`*`aProcName`*`, `*`aProcParm`*`)`
Argument list:	*`aProcName`*	The name of a stored procedure to call
	`aProcParm`	A parameter set to pass to the stored procedure

See also:	`ResultSet` object

database.storedProcArgs() (Method)

Creates a prototype for a stored procedure and controls the argument passing.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 3.0	
Property/method value type:	`Stproc` object	
JavaScript syntax:	NES	`database.storedProcArgs(`*`aProc`*`)`
	NES	`database.storedProcArgs(`*`aProc`*`, `*`someArgTypes`*`)`

Argument list:	*aProc*	The name of a stored procedure
	someArgTypes	Some argument types to define the API

The prototype stored procedure object supports input and output parameters. This prototype object is used to indicate the direction of values in these parameters.

See also:	`DbPool.storedProcArgs()`

database.toString() (Method)

Returns a string equivalent of the database object.

Availability:	JavaScript – 1.1 Netscape Enterprise Server – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`database.toString()`

The value of the object is converted to a string value that represents its value.

See also:	`DbPool.toString()`

dataTransfer object (Object/JScript)

An object used during drag and drop operations to provide access to data being dragged.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myDataTransfer* = *myEvent*.dataTransfer
Object properties:	`dropEffect, effectAllowed`	
Object methods:	`clearData(), getData(), setData()`	

This also assists with access to the clipboard while items are being dragged and dropped. You need to access this object via the `dataTransfer` property of the `event` object.

Operating this functionality is quite complex and you should check out the various examples covered in the documentation at the Microsoft developer web site.

See also:	`Event.dataTransfer`

Property	JavaScript	JScript	N	IE	Opera	Notes
`dropEffect`	-	5.0 +	-	5.0 +	-	-
`effectAllowed`	-	5.0 +	-	5.0 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`clearData()`	-	5.0 +	-	5.0 +	-	-
`getData()`	-	5.0 +	-	5.0 +	-	-
`setData()`	-	5.0 +	-	5.0 +	-	-

dataTransfer.clearData() (Method)

Clears any data currently in the `transfer` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myDataTransfer*`.clearData()`

The contents of the `dataTransfer` object are discarded according to the values specified. To understand what this means requires a short discussion about clipboards and how they work.

The Microsoft Windows clipboard is modelled very closely on the clipboard mechanisms originally implemented in the Macintosh operating system in 1984, when it was first released. The Macintosh environment is constructed around a set of resource objects. When a selection is cut or copied to the clipboard, what is actually stored is a set of resources that describe the selection in a variety of different ways. This enables the target application that receives a paste command to select the most appropriate piece of data that it can understand.

For example, in a music application, the selected region of notation is copied to the clipboard as the following:

- Native application specific data
- A MIDI sequence
- A picture of the notation on screen

If you then goto a word processor and paste in the clipping, it won't understand the MIDI data or the application specific data. However, it could very well understand the picture data and would choose that as being the most appropriate.

This explains how data is lost when you cut and paste between several applications and back to the originating application. Some applications are quite good at preserving the entire contents of the clipboard even though they may only understand one of its component varieties. An example of that is the Macintosh Scrapbook application. It can only display a picture when music clippings are pasted in from a MIDI sequencer. However, it preserves all the component resources and if you copy an item from the scrapbook, you get the complete set again.

Going back to the MSIE browser and the `dataTransfer` object, the drag drop and clipboard mechanism carries a collection of related resources. Each is a copy of the source item in different forms. This `clearData()` method can be used to discard the forms that you no longer need.

It accepts the following values in its argument:

- Text
- URL
- File
- HTML
- Image

You can specify one or several of these to discard the values you no longer need.

This method should be called in an `onDragStart` event handler for maximum effect. If you need to override the default behavior of the target receiver, then you may want to use this in the `onDrop` event handler too.

See also:	`clipboardData` object, `onDragStart`, `onDrop`

dataTransfer.dropEffect (Property)

Set the effect when the element is dropped.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDataTransfer`*`.dropEffect`

This property defines the effect when the data item is dropped onto the receiving element. It needs to be used with the `effectAllowed` property. Controlling the drag drop operation at this level is quite complicated and you should consult the Microsoft developer web site for an in depth explanation of how it works.

This property accepts the following values:

- `copy`
- `link`
- `move`
- `none`

This value can be set during certain event handlers, during the drag operation. These events are likely candidates:

- `onDragEnter`
- `onDragOver`
- `onDrop`

See also:	`dataTransfer.effectAllowed`, `onDragEnter`, `onDragOver`, `onDrop`

Web-references:

http://msdn.microsoft.com/workshop/author/dhtml/reference/properties/dropeffect.asp

dataTransfer.effectAllowed (Property)

Indicates whether the drop effect is allowed or not.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myDataTRansfer`*`.effectAllowed`

You should set this property in the `onDragStart` event handler.

The following values can be assigned to this property, to control whether the drag drop is allowed to happen:

- copy
- link
- move
- copyLink
- copyMove
- linkMove
- all
- none
- uninitialized

Assigning values to this property must be done bearing in mind what you have assigned to the `dropEffect` property as the two work together.

See also:	`dataTransfer.dropEffect`, `onDragStart`

dataTransfer.getData() (Method)

Gets the data from the `transfer` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDataTRansfer`*`.getData()`

This is used to extract the data via the `dataTransfer`, or `clipboardData` objects. The value returned will be in the form of a string and will either be the textual content or a URL reference to a non-text item.

This mode of access preserves the various security needs when accessing values from one frame to another.

This method is most useful within the `onCopy` and `onCut` event handlers.

See also:	`clipboardData` object, `onCopy`, `onCut`

dataTransfer.setData() (Method)

Sets the data in the `transfer` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myDataTransfer*`.setData(`*aType, aString*`)`
Argument list:	*aType*	The value `TEXT` or `URL` to indicate what kind of data is in aString
	aString	A text string to be added to the transfer object

You would use this method to put data into the clipboard. This is done by accessing the `clipboardData` or `dataTransfer` objects during an event handler. You can only store textual values into the clipboard with this method. However they may be the raw text data itself or a description of the URL where non-text data can be found.

The method takes two arguments. The first can be set to the value "`TEXT`" or "`URL`" to indicate which kind of data is being stored. The second argument is the data to be added to the `transfer` object.

This method returns a Boolean value indicating success or failure of the `setData()` call.

Date and time (Definition)

There is a variety of ways to work with dates and times.

Date and time values are manipulated by means of a `Date` object. You can instantiate a new copy of the built-in `Date` object with its constructor. The new instance will hold a fixed time value, which might be 'the time now' or may have been initialized with some other value.

You can get and set various components of the time and date values for a `Date` object. Note that, although you can use the set methods to set a `Date` object to the system time, this will not alter the system time of the machine you are running the script in.

It is possible that some implementations may provide that capability, but it is not defined as part of the standard.

See also:	Broken down time, Calendar time, Daylight savings time adjustment, Local time, Localization, Universal coordinated time

Date constant (Definition)

A constant date value.

Date constants are Date objects with predefined values. These need to be manufactured from component parts and parsed by the `Date` object constructor.

See also:	Constant expression, `Date()` constructor, `Date()` function, `Date.constructor`, `Date.parse()`

Date from time (Time calculation)

A date and time algorithm defined by ECMAScript.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

ECMA compliant implementations use the extended Gregorian system for dates. These are all based on 01-January-1970 UTC as a starting point. Although the date handling in JavaScript is flexible and generally comprehensive, there may be additional date computations required in some implementations.

The formula for calculating day number is shown here:

```
t = an instant in time measured in milliseconds relative to 01-January-1970 UTC.

msPerDay = 86400000

Day(t) = floor(t/msPerDay)
```

All non-leap years have 365 days with the usual number of days in each month. Leap years have an extra day in February. The calculation shown below uses known leap years and non-leap years to adjust the day numbers, and yield the day number of the first day of the given year and then use that to work out the time in milliseconds from when the year started:

```
DayFromYear(y) =

365 * (y - 1970) +

floor((y - 1969) / 4) -

floor((y - 1901) / 100) +

floor((y - 1601) / 400)

TimeFromYear(y) = msPerDay * DayFromYear(y)

YearFromTime(t) = The largest integer y to make TimeFromYear(y) less than or equal
to t.

DayWithinYear(t) = Day(t) - DayFromYear(YearFromTime(t))
```

The month value is worked out with this formulaic framework:

`MonthFromTime(t) = lookup` according to `DayWithinYear(t)` falling into a range according to the following table:

Greater than	Less than	Month	Name
000	031	0	January
031	059 + InLeapYear(t)	1	February
059 + InLeapYear(t)	090 + InLeapYear(t)	2	March
090 + InLeapYear(t)	120 + InLeapYear(t)	3	April
120 + InLeapYear(t)	151 + InLeapYear(t)	4	May
151 + InLeapYear(t)	181 + InLeapYear(t)	5	June
181 + InLeapYear(t)	212 + InLeapYear(t)	6	July
212 + InLeapYear(t)	243 + InLeapYear(t)	7	August
243 + InLeapYear(t)	273 + InLeapYear(t)	8	September
273 + InLeapYear(t)	304 + InLeapYear(t)	9	October
304 + InLeapYear(t)	334 + InLeapYear(t)	10	November
334 + InLeapYear(t)	365 + InLeapYear(t)	11	December

The date within the month is worked out in a similar way, and could probably share a common table of month lengths or offsets.

`DateFromTime(t) = lookup` according to `MonthFromTime(t)` selecting a value to subtract according to the following table:

Month value	Subtract this
00	-1
01	30
02	58 + InLeapYear(t)
03	89 + InLeapYear(t)
04	119 + InLeapYear(t)
05	150 + InLeapYear(t)
06	180 + InLeapYear(t)
07	211 + InLeapYear(t)
08	242 + InLeapYear(t)
09	272 + InLeapYear(t)
10	303 + InLeapYear(t)
11	333 + InLeapYear(t)

See also:	Broken down time, Date number, Day from year, Day number, Day within year, Month from time, Time from year, Time range, Year from time

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.5

ECMA 262 edition 3 – section – 15.9.1.5

Date number (Time calculation)

A date and time algorithm defined by ECMAScript.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

A date number is identified by an integer in the range 1 through 31, inclusive. The specific range depends on the month being considered and whether a leap year is in force.

See also:	Broken down time, Date from time, Day from year, Day number, Day within year, Month from time, Time from year, Time range, Year from time

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.5

ECMA 262 edition 3 – section – 15.9.1.5

Date object (Object/core)

An object of the class "`Date`".

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	*myDate* = Date
	N	*myDate* = *myEvent*.timeStamp
	N	*myDate* = *myMouseEvent*.timeStamp
	N	*myDate* = *myMutationEvent*.timeStamp
	N	*myDate* = *myUIEvent*.timeStamp
	-	*myDate* = new Date()
Object properties:	constructor, length, prototype	
Class methods:	parse(), UTC()	

Object methods:	`getDate()`, `getDay()`, `getFullYear()`, `getHours()`, `getMilliseconds()`, `getMinutes()`, `getMonth()`, `getSeconds()`, `getTime()`, `getTimezoneOffset()`, `getUTCDate()`, `getUTCDay()`, `getUTCFullYear()`, `getUTCHours()`, `getUTCMilliseconds()`, `getUTCMinutes()`, `getUTCMonth()`, `getUTCSeconds()`, `getVarDate()`, `getYear()`, `parse()`, `setDate()`, `setFullYear()`, `setHours()`, `setMilliseconds()`, `setMinutes()`, `setMonth()`, `setSeconds()`, `setTime()`, `setUTCDate()`, `setUTCFullYear()`, `setUTCHours()`, `setUTCMilliseconds()`, `setUTCMinutes()`, `setUTCMonth()`, `setUTCSeconds()`, `setYear()`, `toDateString()`, `toGMTString()`, `toLocaleDateString()`, `toLocaleString()`, `toLocaleTimeString()`, `toSource()`, `toString()`, `toTimeString()`, `toUTCString()`, `valueOf()`

A `Date` object contains a number that denotes a particular instant in time that is accurate to within a millisecond. The number value may also contain `NaN`, which indicates that the `Date` object does not represent a valid instant in time.

The prototype for the `Date` prototype object is the `Object` prototype object.

Instances of the `Date` object have no special properties beyond those they inherit from the `Date.prototype` object.

JavaScript version 1.2 and the ECMAScript standard both mandate additional methods that the `Date` object should support. These are generally useful when computing year numbers higher than 1999.

Warnings:

- The `Date` object is particularly bug prone in Netscape 2. If this browser version is important, you may need to provide significant amounts of date correcting logic or avoid the use of date values altogether.

Example code:

```
<!-- Display time since document loaded --->
<HTML>
<HEAD>
<SCRIPT>
window.myDate1 = new Date();
</SCRIPT>
</HEAD>
<BODY>
<DIV ID="TEXTCELL">
0000
</DIV>
<FORM>
<INPUT TYPE="button" VALUE="CLICK ME" onClick="clickMe()">
</FORM>
<SCRIPT>
function clickMe()
{
   myDate2 = new Date();
```

```
    myDelta = myDate2 - window.myDate1;
    document.all.TEXTCELL.innerText = myDelta/1000;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Broken down time, Browser version compatibility, Calendar time, Compatibility strategies, `Date.Class`, `Date.length`, `Date.prototype`, `Event.timeStamp`, `java.util.Date`, JellyScript, `MakeDate()`, `MakeDay()`, `MakeTime()`, `Native` object, `Object` object, Time range, Time value, `TimeClip()`, `unwatch()`, `watch()`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`constructor`	1.1 +	1.0 +	3.0 +	3.02 +	-	-	2 +	-
`length`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	ReadOnly, DontEnum.
`prototype`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	ReadOnly, DontDelete, DontEnum.

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`getDate()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`getDay()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`getFullYear()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getHours()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`getMilliseconds()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getMinutes()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`getMonth()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`getSeconds()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`getTime()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`getTimezoneOffset()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	Warning
`getUTCDate()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getUTCDay()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getUTCFullYear()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	Warning
`getUTCHours()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getUTCMilliseconds()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getUTCMinutes()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getUTCMonth()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getUTCSeconds()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`getVarDate()`	-	3.0 +	-	4.0 +	-	-	-	-
`getYear()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	Warning, Deprecated

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`parse()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`setDate()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`setFullYear()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setHours()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`setMilliseconds()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setMinutes()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`setMonth()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`setSeconds()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`setTime()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`setUTCDate()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setUTCFullYear()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setUTCHours()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setUTCMilliseconds()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setUTCMinutes()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setUTCMonth()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setUTCSeconds()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`setYear()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	Warning, Deprecated
`toDateString()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toGMTString()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	Warning, Deprecated
`toLocaleDateString()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toLocaleString()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`toLocaleTimeString()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toSource()`	1.3 +	3.0 +	4.06 +	4.0 +	-	-	3 +	-
`toString()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	Warning
`toTimeString()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toUTCString()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	2 +	-
`valueOf()`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	2 +	-

Cross-references:

ECMA 262 edition 2 – section – 10.1.5

ECMA 262 edition 2 – section – 15.9

ECMA 262 edition 2 – section – 15.9.6

ECMA 262 edition 3 – section – 10.1.5

ECMA 262 edition 3 – section – 15.9

O'Reilly *JavaScript Definitive Guide* – page – 48

Date() (Constructor)

A `Date` object constructor.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0	
Property/method value type:	`Date` object	
JavaScript syntax:	-	`new Date()`
	-	`new Date(`*`aValue`*`)`
	-	`new Date(`*`aYear`*`, `*`aMonth`*`)`
	-	`new Date(`*`aYear`*`, `*`aMonth`*`, `*`aDate`*`)`
	-	`new Date(`*`aYear`*`, `*`aMonth`*`, `*`aDate`*`, `*`anHour`*`)`
JavaScript syntax:	-	`new Date(`*`aYear`*`, `*`aMonth`*`, `*`aDate`*`, `*`anHour`*`, `*`aMinute`*`)`
	-	`new Date(`*`aYear`*`, `*`aMonth`*`, `*`aDate`*`, `*`anHour`*`, `*`aMinute`*`, `*`aSecond`*`)`
	-	`new Date(`*`aYear`*`, `*`aMonth`*`, `*`aDate`*`, `*`anHour`*`, `*`aMinute`*`, `*`aSecond`*`, `*`aMillisecond`*`)`
Argument list:	*`aDate`*	An optional date within the month value
	`aMillisecond`	An optional value between 0 and 999 milliseconds
	`aMinute`	An optional value between 0 and 59 minutes
	`aMonth`	An optional 0 to 11 month value
	`anHour`	A value between 0 and 23 hours
	`aSecond`	An optional value between 0 and 59 seconds
	`aYear`	A full year value
	`aValue`	A time in UTC milliseconds

The result of calling this constructor is a `date` object with the indicated date and time value.

Calling the `Date()` constructor with the new operator creates a fresh object based on the `Date` prototype. The value of this new `Date` object depends on the parameters specified when the constructor was invoked.

This is not the same as simply calling the `Date()` function which would yield the current system date and time at the instant it was called.

The arguments to the `Date()` constructor are all optional, but they are also positional. This means that you must mark empty positions with comma separated null values to indicate that a parameter needs to be skipped. The time values are assumed to be measured in local time and not UTC.

The prototype of the new `Date` object is the built in `Date` prototype object.

Functionally, the algorithm that manufactures a new date value uses the internal `MakeDay()`, `MakeTime()`, and `MakeDate()` functions that we describe elsewhere.

If the year value is less than 99, then the date creation adds 1900 to it and assumes the date is in the 20th century. To avoid millennium problems, always specify a full year number.

The following rules apply where items are omitted from the right of the argument list.

Zero values are assumed for hours, minutes and seconds. When all three are missing, the time is assumed to be midnight.

The date value is assumed to be the first of the month, and the default month is not considered since a single value on its own is taken to mean a millisecond time value in UTC time coordinates.

When all arguments are omitted, the time value for the new object is set to the current time in UTC time coordinates.

Putting null values in place of the year month and date sets the time correctly but unpredictable date values are substituted.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define the date but assume the time is set to zero
// Note that month numbers start at zero
myDate1 = new Date(1954, 0, 19);
document.write(myDate1);
document.write("<BR>");
// Define the time in minutes but make up any old date
myDate2 = new Date(null, null, null, 12, 14);
document.write(myDate2);
document.write("<BR>");
// Fully qualified date value
myDate3 = new Date(1984, 9, 23, 1, 0, 0, 0);
document.write(myDate3);
document.write("<BR>");
// Output message of the day
</SCRIPT>
</BODY>
</HTML>
```

See also: Constructor function, `constructor` property, Date constant, `Date()` function, `Date.UTC()`, `Global` object, `MakeDate()`, `MakeDay()`, `MakeTime()`, new, Object constant, `TimeClip()`

Cross-references:

ECMA 262 edition 2 – section – 15.1.3.7

ECMA 262 edition 2 – section – 15.9.3

ECMA 262 edition 2 – section – 15.9.4

ECMA 262 edition 3 – section – 15.9.3

Date() (Function)

A function that returns the current date.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0
Property/method value type:	`String primitive`
JavaScript syntax:	- `Date()`

This function returns a string primitive representing the current UTC time value.

When the `Date()` constructor is called as a function rather than in a new expression, it returns a string representing the current time (in UTC time). Note that when calling it as a function the arguments are all ignored and it is not equivalent to calling the `Date()` constructor in a new expression at all.

Effectively, the function call to a `Date()` constructor behaves as if you had coded this fragment of JavaScript in the script source text:

```
(new Date()).toString()
```

See also:	Cast operator, Constructor function, `constructor` property, Date constant, `Date()` constructor, Implicit conversion

Cross-references:

ECMA 262 edition 2 – section – 15.1.3.7

ECMA 262 edition 2 – section – 15.9.2

ECMA 262 edition 3 – section – 15.9.2

Date.Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition – 2

This is an internal property that describes the class that a `Date` object instance is a member of. The reserved words suggest that, in the future, this property may be externalized.

See also:	Class, `Date` object

Property attributes:

DontEnum, Internal.

Cross-references:

ECMA 262 edition 2 – section – 8.6.2

ECMA 262 edition 2 – section – 15.9.3.1

ECMA 262 edition 3 – section – 8.6.2

Date.constructor (Property)

A reference to a constructor object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0
Property/method value type:	`Date constructor object`
JavaScript syntax:	- `Date.constructor`

The initial value of the date prototype constructor is the built-in `Date` constructor.

You can use this as one way of creating `Date` objects although it is more popular to use the new `Date()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	Date constant, `Date.length`, `Date.parse()`, `Date.prototype`, `Date.UTC()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.1

ECMA 262 edition 3 – section – 15.9.3

Date.getDate() (Method)

Returns the day number within a month for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myDate*`.getDate()`

This method returns a day number within a month from the date value of the receiving object. The value is 1 based and will not exceed 31 although the range is variable based on month and leap year contexts.

The date is computed according to local time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
// Create diary of special days per month
myDiary = new Array();
myDiary[1]  = "Pay into bank account";
myDiary[28] = "Invoice customers";
myDiary[14] = "Chase customers for payment";
myDiary[5]  = "Top up fuel in generator";
// Work out day number
myDate = new Date();
myDay = myDate.getDate();
// Output message of the day
document.write(myDiary[myDay]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype,Date.setDate()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.10

ECMA 262 edition 3 – section – 15.9.5.14

Date.getDay() (Method)

Returns the weekday number for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.getDay()

A weekday number for the date value of the receiving object is returned. The value will be in the range 0 to 6, with 0 representing Sunday.

The value is computed according to Local time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
myDate = new Date();
document.write(myArray[myDate.getDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.12

ECMA 262 edition 3 – section – 15.9.5.16

Date.getFullYear() (Method)

Returns the full year for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myDate*`.getFullYear()`

The full year number of the date value in the receiving object is returned as a four digit value. This is much preferred to the get`Year()` method which suffers from Y2K ambiguities and is now deprecated in favor of this method.

The year number is a local time value.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
// Calculate which century from the year number
myDate = new Date();
myCentury = 1 + myDate.getFullYear()/100;
switch (myCentury % 10)
{
case 1 : mySuffix = "st";
break;

case 2 : mySuffix = "nd";
break;

case 3 : mySuffix = "rd";
break;

default : mySuffix = "th";
break;

}
document.write(myCentury + mySuffix + " century");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getYear(),Date.prototype,Date.setFullYear(), Date.setUTCFullYear(),Date.setYear()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.6

ECMA 262 edition 3 – section – 15.9.5.10

Date.getHours() (Method)

Returns the hour value for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.getHours()

The hours of the time value contained in the receiving object. This is returned as an integer in the range 0 to 23 inclusive.

The value is computed according to local time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
// Generate a flag based on work or play time
myDate = new Date();
myHours = myDate.getHours();
if((myHours > 8) && (myHours < 17))
{
document.write("At work");
}
else
{
document.write("At play");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype, Date.setHours(), Date.setUTCHours()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.14

ECMA 262 edition 3 – section – 15.9.5.18

Date.getMilliseconds() (Method)

Return the milliseconds value of a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.getMilliseconds()

The milliseconds of the time value contained in the receiving object. The result will be an integer in the range 0 to 999 inclusive.

The value is computed according to local time coordinates.

In the example a timer is set up to trigger a function call every 1000 milliseconds. However, processing load may affect how accurately the timer is executed. By measuring the time on each call, we know that the millisecond value should be the same each time but you will probably observe a small and random perturbation in the value. You can compensate for this by making a small adjustment to the timer value to take this error into account. You will most likely need to reduce the value 1000 by a few milliseconds to get a more accurate timer.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<DIV ID="RESULT">000</DIV>
<SCRIPT>
// Check the accuracy of a timer by printing the milliseconds
setInterval("monitor()", 1000);
myOldMSecs = 000;
function monitor()
{
   myDate  = new Date();
   myMsecs = myDate.getMilliseconds();
   myErr = myMsecs - myOldMSecs;
   document.all.RESULT.innerText = myMsecs + " (" + myErr + " milliseconds of
error)";
   myOldMSecs = myMsecs;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype, Date.setMilliseconds(), Date.setUTCMilliseconds()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.20

ECMA 262 edition 3 – section – 15.9.5.24

Date.getMinutes() (Method)

Returns the minutes value of a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myDate*`.getMinutes()`

The minutes of the time value contained in the receiving object. The result will be an integer in the range 0 to 59 inclusive.

The value is computed according to local time coordinates.

Example code:

```
<!-- A bar chart indicating minutes of the hour -->
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
<TR HEIGHT=10>
<TD ID="LEFT"  WIDTH=100 STYLE="background-color:RED"> </TD>
<TD ID="RIGHT" WIDTH=500 STYLE="background-color:BLUE"> </TD>
</TR>
</TABLE>
<SCRIPT>
setBar()
setInterval("setBar()", 60000);
function setBar()
{
   myDate    = new Date();
   myMinutes = myDate.getMinutes();
   myOther   = 60 - myMinutes;
   document.all.LEFT.style.width  = 1+myMinutes*10;
   document.all.RIGHT.style.width = 1+myOther*10;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`, `Date.setMinutes()`, `Date.setUTCMinutes()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.16

ECMA 262 edition 3 – section – 15.9.5.20

Date.getMonth() (Method)

Returns the month value of a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myDate*`.getMonth()`

The month number for the receiving date object is returned. This is a zero based value in the range 0 to 11 inclusive. The value 0 represents January, and 11 represents December.

The value is in local time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate  = new Date();
myMonth = myDate.getMonth();
egyptianCalendar(myMonth);
// Generate attributes of the egyptian calendar based on month
function egyptianCalendar(aMonth)
{
   mySeasonArray = new Array("Akhet", "Peret", "Shemu");
   myMonthArray  = new Array("I", "II", "III", "IV");
   myNameArray   = new Array("Thoth", "Phaophi", "Athyr", "Choiak", "Tybi",
"Mechir",       "Phamenoth", "Pharmuthi", "Pachons", "Payni", "Epiphi", "Mesore");
   myDeityArray  = new Array("Tekhi", "Ptah-aneb-res-f", "Het-hert", "Sekhet",
"Amsu",           "Rekeh-ur", "Rekeh-netches", "Rennutet", "Khensu", "Kenthi",
"Apt", "Heru-khuti");
   myTypeArray   = new Array("F", "M", "F", "F", "M", "M", "M", "F", "M", "M",
"F", "M");
   mySeason = mySeasonArray[Math.floor(aMonth/4)];
   myMonth  = myMonthArray[aMonth%4];
   myName   = myNameArray[aMonth];
   myDeity  = myDeityArray[aMonth];
   switch(myTypeArray[aMonth])
```

```
        {
           case "F" : myType = "Goddess";
              break;
           case "M" : myType = "God";
              break;
        }
     document.write("Index : "  + aMonth   + "<BR>");
     document.write("Season : " + mySeason + "<BR>");
     document.write("Month : "  + myMonth  + "<BR>");
     document.write("Name : "   + myName   + "<BR>");
     document.write("Deity : "  + myDeity  + "<BR>");
     document.write("Type : "   + myType   + "<BR>");
  }
  </SCRIPT>
  </BODY>
  </HTML>
```

See also:	`Date.prototype,Date.setMonth(),Date.setUTCMonth()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.8

ECMA 262 edition 3 – section – 15.9.5.12

Date.getSeconds() (Method)

Returns the seconds value of a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*`.getSconds()`

The seconds of the time value contained in the receiving object. The result will be an integer in the range 0 to 59 inclusive.

The value is computed according to UTC time coordinates.

Example code:

```
<!-- A bar chart indicating seconds of the minute -->
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
<TR HEIGHT=10>
<TD ID="LEFT"  WIDTH=100 STYLE="background-color:RED"> </TD>
<TD ID="RIGHT" WIDTH=500 STYLE="background-color:BLUE"> </TD>
</TR>
</TABLE>
<SCRIPT>
setBar()
setInterval("setBar()", 990);
function setBar()
{
myDate    = new Date();
mySeconds = myDate.getSeconds();
myOther   = 60 - mySeconds;
document.all.LEFT.style.width  = 1+mySeconds*10;
document.all.RIGHT.style.width = 1+myOther*10;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`, `Date.setSeconds()`, `Date.setUTCSeconds()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.18

ECMA 262 edition 3 – section – 15.9.5.22

Date.getTime() (Method)

Returns the time value of a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Time value`
JavaScript syntax:	- *myDate*`.getTime()`

The time value of the receiving object. This will be an integer that represents the time in milliseconds since the time origin at midnight on January the first, 1970.

This method returns the time value of the receiving object.

Example code:

```
<!-- Hand and eye coordination and reaction timer -->
<HTML>
<HEAD></HEAD>
<BODY>
<DIV ID="ONE" STYLE="background-color:YELLOW">
Click button 1 then button two as quickly as you can
</DIV>
<FORM>
<INPUT TYPE="button" VALUE="1" onClick="clickMe1()">
<INPUT TYPE="button" VALUE="2" onClick="clickMe2()">
</FORM>
<SCRIPT>
function clickMe1()
{
myDate1 = new Date();
myTime1 = myDate1.getTime();
}
function clickMe2()
{
myDate2 = new Date();
myTime2 = myDate2.getTime();
document.all.ONE.innerText = (myTime2 - myTime1) + " Milliseconds";
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`, `Date.setTime()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.4

ECMA 262 edition 3 – section – 15.9.5.9

Date.getTimezoneOffset() (Method)

Returns the time zone offset for the date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*`.getTimezoneOffset()`

The difference between UTC time and local time measured in minutes.

The result of this method is a zero based integer equal to the time difference between GM reference time and the clock in the client system. The value is measured in minutes and will be in the range 0 to 59 inclusive.

Warnings:

- Note that international time zones are generally in 60-minute intervals. However, a very few are less than 60 minutes.
- There are deficiencies in the accuracy of this method in Netscape 2 and 3. You should test whether your scripts work properly on those browsers if backwards compatibility is important to your project.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myTZO = myDate.getTimezoneOffset();
if(myTZO == 0)
{
   document.write("Same time as London, England.");
}
else
{
   document.write(myTZO + " hours relative to London, England time.");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.22

ECMA 262 edition 3 – section – 15.9.5.26

Date.getUTCDate() (Method)

Returns the UTC day of month value for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.getUTCDate()

This method returns a day number within a month from the date value of the receiving object. The value is 1 based and will not exceed 31 although the range is variable based on month and leap year contexts.

The date is computed according to UTC time coordinates.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Create diary of special days per month
myDiary = new Array();
myDiary[1]  = "Pay into bank account";
myDiary[28] = "Invoice customers";
myDiary[14] = "Chase customers for payment";
myDiary[5]  = "Top up fuel in generator";
// Work out day number
myDate = new Date();
myDay = myDate.getUTCDate();
// Output message of the day
document.write(myDiary[myDay]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype, Date.setUTCDate()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.11

ECMA 262 edition 3 – section – 15.9.5.15

Date.getUTCDay() (Method)

Returns the UTC weekday number for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.getUTCDay()

The result of this method is a weekday number for the date value of the receiving object. The value will be in the range 0 to 6 inclusive. The 0 value represents Sunday.

The value is computed according to UTC time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
myDate = new Date();
document.write(myArray[myDate.getUTCDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.13

ECMA 262 edition 3 – section – 15.9.5.17

Date.getUTCFullYear() (Method)

Returns the UTC full year value for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.getUTCFullYear()

The year number for the receiving object normalized to UTC time coordinates. The result is a full 4 digit year number value based on the UTC time value.

Warnings:

- If you are computing relative times or developing scripts that operate on very early dates, Netscape will not support negative year numbers, and cannot go any earlier than year zero. MSIE can, if necessary return a negative year number.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
// Calculate which century from the year number
myDate = new Date();
myCentury = 1 + myDate.getUTCFullYear()/100;
switch (myCentury % 10)
{
case 1 : mySuffix = "st";
break;

case 2 : mySuffix = "nd";
break;

case 3 : mySuffix = "rd";
break;

default : mySuffix = "th";
break;

}
document.write(myCentury + mySuffix + " century");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`, `Date.setUTCFullYear()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.7

ECMA 262 edition 3 – section – 15.9.5.11

Date.getUTCHours() (Method)

Returns the UTC hours value for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- `myDate.getUTCHours()`

The result of this method is the hours component of the time value contained in the receiving object. The value will be in the range 0 to 23 inclusive.

The value is computed according to UTC time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
// Generate a flag based on work or play time
myDate = new Date();
myHours = myDate.getUTCHours();
if((myHours > 8) && (myHours < 17))
{
document.write("At work");
}
else
{
document.write("At play");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`, `Date.setUTCHours()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.15

ECMA 262 edition 3 – section – 15.9.5.19

Date.getUTCMilliseconds() (Method)

Returns the UTC milliseconds for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.getUTCMilliseconds()

The result of this method is the milliseconds of the time value contained in the receiving object. The value will be in the range 0 to 999 inclusive.

The value is computed according to UTC time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<DIV ID="RESULT">000</DIV>
<SCRIPT>
// Check the accuracy of a timer by printing the milliseconds
setInterval("monitor()", 1000);
myOldMSecs = 000;
function monitor()
{
myDate  = new Date();
myMsecs = myDate.getUTCMilliseconds();
myErr = myMsecs - myOldMSecs;
document.all.RESULT.innerText = myMsecs + " (" + myErr + " milliseconds of
error)";
myOldMSecs = myMsecs;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`, `Date.setUTCMilliseconds()`, `Event.timeStamp`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.21

ECMA 262 edition 3 – section – 15.9.5.25

Date.getUTCMinutes() (Method)

Returns the UTC minutes value for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- `myDate.getUTCMinutes()`

The result is the minutes of the time value contained in the receiving object. The value will be in the range 0 to 59 inclusive.

The value is computed according to UTC time coordinates.

Example code:

```
<!-- A bar chart indicating minutes of the hour -->
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
<TR HEIGHT=10>
<TD ID="LEFT"  WIDTH=100 STYLE="background-color:RED"> </TD>
<TD ID="RIGHT" WIDTH=500 STYLE="background-color:BLUE"> </TD>
</TR>
</TABLE>
<SCRIPT>
setBar()
setInterval("setBar()", 60000);
function setBar()
{
myDate    = new Date();
myMinutes = myDate.getUTCMinutes();
myOther   = 60 - myMinutes;
document.all.LEFT.style.width  = 1+myMinutes*10;
document.all.RIGHT.style.width = 1+myOther*10;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype,Date.setUTCMinutes()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.17

ECMA 262 edition 3 – section – 15.9.5.21

Date.getUTCMonth() (Method)

Returns the UTC month number for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myDate*.getUTCMonth()

The result of this method is month number for the receiving `date` object. The value will be in the range 0 to 11 inclusive, with the value 0 representing January, and 11 representing December.

The month number is measured in UTC time coordinates.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate  = new Date();
myMonth = myDate.getUTCMonth();
egyptianSeasons(myMonth);
// Generate attributes of the egyptian calendar based on month
function egyptianSeasons(aMonth)
{
mySeasonArray = new Array("Akhet", "Peret", "Shemu");
myMonthArray  = new Array("I", "II", "III", "IV");
mySeason = mySeasonArray[Math.floor(aMonth/4)];
myMonth  = myMonthArray[aMonth%4];
document.write("Index : "  + aMonth   + "<BR>");
document.write("Season : " + mySeason + "<BR>");
document.write("Month : "  + myMonth  + "<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype`, `Date.setUTCMonth()`

Cross-references:

ECMA 262 edition 2 – section – 15.5.9.9

ECMA 262 edition 3 – section – 15.9.5.13

Date.getUTCSeconds() (Method)

Returns the UTC seconds value for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myDate*`.getUTCSeconds()`

The result of this method is the seconds part of the time value contained in the receiving object. The value will be in the range 0 to 59 inclusive.

The value is computed according to UTC time coordinates.

Example code:

```
<!-- A bar chart indicating seconds of the minute -->
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
<TR HEIGHT=10>
<TD ID="LEFT"  WIDTH=100 STYLE="background-color:RED"> </TD>
<TD ID="RIGHT" WIDTH=500 STYLE="background-color:BLUE"> </TD>
</TR>
</TABLE>
<SCRIPT>
setBar()
setInterval("setBar()", 990);
function setBar()
{
myDate    = new Date();
mySeconds = myDate.getUTCSeconds();
myOther   = 60 - mySeconds;
document.all.LEFT.style.width  = 1+mySeconds*10;
document.all.RIGHT.style.width = 1+myOther*10;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.prototype,Date.setUTCSeconds()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.19

ECMA 262 edition 3 – section – 15.9.5.23

Date.getVarDate() (Method)

A special date format for use with ActiveX objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`VarDate value`	
JavaScript syntax:	IE	*`myDate`*`.getVarDate()`

This method is provided so that you can obtain a special date format called `VT_DATE`. This is used for communicating with ActiveX objects on the Windows platform.

This method is not standardized in ECMAScript and is not supported in Netscape, so you should use it with great caution. In any case, if you are likely to need it, then you'll likely be using an `ActiveX` object that is only available on the Windows platform.

Date.getYear() (Method)

Returns a 2 digit non-Y2K compliant year for a date/time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0 Deprecated from JavaScript 1.3	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myDate`*`.getYear()`

The result of this method is a 2 digit year number for the date contained in the receiving object.

The year number is a local time value.

However, in this case, the value 1900 is subtracted from the year on the assumption that a two digit year will result.

This can lead to year 2000 problems as the year value may be modulo 100 or it may carry over.

This is why the method is deprecated and should be replaced by `Date.getFullYear()` in your scripts.

As of ECMA edition 3 it is no longer included in the standard although implementations may still provide it for backwards compatibility.

Warnings:

- Although it is described in the ECMA standard, it is noted that this function is not formally part of the standard and an ECMA compliant implementation need not provide it. Instead, the `Date.getFullYear()` function should be used.
- Nevertheless, some implementations may support this function although it is strongly recommended that you avoid its use.

See also:	`Date.getFullYear()`,`Date.prototype`, `Date.setFullYear()`,`Date.setYear()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.5

ECMA 262 edition 3 – section – B.2.4

Date.length (Property)

The length of a `date` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*.length

The `length` property for a `Date` object returns the maximum number of arguments that the `Date()` constructor function accepts.

The length property always returns the value 7 for a `Date` object.

See also:	`Date` object, `Date.constructor`

Property attributes:

ReadOnly, DontEnum.

Cross-references:

ECMA 262 edition 2 – section – 15.9.4

ECMA 262 edition 3 – section – 15.9.4

Date.parse() (Method/static)

A class based factory method for converting strings to `Date` objects.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`UTC time value`
JavaScript syntax:	- `Date.parse(`*`aString`*`)`
Argument list:	*`aString`* A string containing a meaningful date value

The argument used with the `Date.parse()` method should process any string containing a sequence of characters that represents a meaningful date value. The `parse()` method then tokenizes that string and converts the value to a number in UTC time coordinates corresponding to the date and time in the string.

The string may be interpreted as a local time, UTC time, or a time in some other time zone depending on the contents of the string.

The result of this method is a UTC time value.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="RESULT1">???</DIV>
<DIV ID="RESULT2">???</DIV>
<FORM>
<HR>
<SELECT ID="IN1">
<OPTION VALUE="Sun">Sunday
<OPTION VALUE="Mon">Monday
<OPTION VALUE="Tue">Tuesday
<OPTION VALUE="Wed">Wednesday
<OPTION VALUE="Thu">Thursday
<OPTION VALUE="Fri">Friday
<OPTION VALUE="Sat">Saturday
</SELECT>
<INPUT ID="IN2" TYPE="text" VALUE="1" SIZE="2">
<SELECT ID="IN3">
<OPTION VALUE="Jan">January
<OPTION VALUE="Feb">February
<OPTION VALUE="Mar">March
<OPTION VALUE="Apr">April
<OPTION VALUE="May">May
<OPTION VALUE="Jun">June
<OPTION VALUE="Jul">July
<OPTION VALUE="Aug">August
```

```
<OPTION VALUE="Sep">September
<OPTION VALUE="Oct">October
<OPTION VALUE="Nov">November
<OPTION VALUE="Dec">December
</SELECT>
<INPUT ID="IN4" TYPE="text" VALUE="2000" SIZE="4">
<INPUT ID="IN5" TYPE="text" VALUE="12" SIZE="2">:
<INPUT ID="IN6" TYPE="text" VALUE="00" SIZE="2">:
<INPUT ID="IN7" TYPE="text" VALUE="00" SIZE="2">
<INPUT ID="IN8" TYPE="text" VALUE="+0000" SIZE="5">
<HR>
<INPUT TYPE="button" VALUE="CLICK ME" onClick="clickMe()">
</FORM>
<SCRIPT>
function clickMe()
{
parseInput  = document.all.IN1.value;
parseInput += " ";
parseInput += document.all.IN2.value;
parseInput += " ";
parseInput += document.all.IN3.value;
parseInput += " ";
parseInput += document.all.IN4.value;
parseInput += " ";
parseInput += document.all.IN5.value;
parseInput += ":";
parseInput += document.all.IN6.value;
parseInput += ":";
parseInput += document.all.IN7.value;
parseInput += " ";
parseInput += document.all.IN8.value;
document.all.RESULT1.innerText = parseInput;

document.all.RESULT2.innerText = Date.parse(parseInput);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Cast operator, Date constant, `Date.constructor`

Cross-references:

ECMA 262 edition 2 – section – 15.9.4.2

ECMA 262 edition 3 – section – 15.9.4.2

Date.prototype (Property)

The prototype for the `Date` object that can be used to extend the interface for all `Date` objects.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Date object`	
JavaScript syntax:	-	`Date.prototype`
	-	*`myDate`*`.constructor.prototype`

The initial value of the prototype property refers to the built-in `Date` prototype object.

The value of the `Date` prototype object is `NaN`.

The example demonstrates how to provide extensions to all instances of this class by adding a function to the prototype object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that extends the output capabilities of Date objects
function millennium()
{
   return this.getFullYear()/1000;
}
// Register the new function
Date.prototype.millennium = millennium;
// Create a date and test the Date.millennium() method
myDate = new Date();
document.write(myDate.millennium())
document.write("<BR>")
</SCRIPT>
</BODY>
</HTML>
```

See also:	`prototype` property

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 – section – 15.2.3.1

ECMA 262 edition 2 – section – 15.9.4.1

ECMA 262 edition 2 – section – 15.9.5

ECMA 262 edition 3 – section – 15.9.4.1

Date.setDate() (Method)

Sets the day number within a month of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Time value`	
JavaScript syntax:	-	*myDate*.setDate(*aDateValue*)
Argument list:	*aDateValue*	A day number within the current month

The result of this method will be the new time value of the containing object having been adjusted by the values passed in as arguments.

The calculations are performed according to local time coordinates.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
myDate = new Date();
myDate.setDate(1);
document.write("The first day of this month is a " + myArray[myDate.getDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getDate(),Date.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.32

ECMA 262 edition 3 – section – 15.9.5.36

Date.setFullYear() (Method)

Sets the full year value of a date/time object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Time value`	
JavaScript syntax:	-	*myDate*.setFullYear(*aYearValue*)
	-	*myDate*.setFullYear(*aYearValue, aMonthValue*)
	-	*myDate*.setFullYear(*aYearValue, aMonthValue, aDateValue*)
Argument list:	*aDateValue*	An optional date within the month value
	aMonthValue	An optional 0 to 11 month value
	aYearValue	A full year value

The result returned by this method will be the new time value of the containing object, having been adjusted by the values passed in as arguments.

The computations are carried out in local time coordinates.

ECMA mandates that the month and date should be optional arguments. If the date value is omitted, the value is provided internally as if the `getDate()` method had been invoked to provide it. Likewise the month value will be replaced by internally calling the `getMonth()` method.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
myDate = new Date();
myDate.setFullYear(myDate.getFullYear() - 1);
document.write("A year ago on today's date, it was a " +
myArray[myDate.getDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getFullYear()`,`Date.getYear()`,`Date.prototype`, `Date.setUTCFullYear()`,`Date.setYear()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.36

ECMA 262 edition 3 – section – 15.9.5.40

Date.setHours() (Method)

Sets the hours of a date/time object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Time value`
JavaScript syntax:	- *myDate*.setHours(anHoursValue) - *myDate*.setHours(*anHoursValue, aMinutesValue*) - *myDate*.setHours(*anHoursValue, aMinutesValue, aSecondsValue*) - *myDate*.setHours(*anHoursValue, aMinutesValue, aSecondsValue, aMillisecondsValue*)

Argument list:		
	aMillisecondsValue	An optional value between 0 and 999 milliseconds
	aMinutesValue	An optional value between 0 and 59 minutes
	anHoursValue	A value between 0 and 23 hours
	aSecondsValue	An optional value between 0 and 59 seconds

The value is computed according to local time coordinates.

ECMA mandates that the minutes, seconds and milliseconds should be optional arguments. If the milliseconds value is omitted, the current milliseconds value is used as if the value had been supplied with a `getMilliseconds()` method. Likewise, the seconds value behaves as if it had been supplied by a `getSeconds()` method, and the minutes as if `getMinutes()` was used, if they are missing.

The result of this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDay1 = myDate.getDay();
for(myEnum=1; myEnum<24; myEnum++)
{
myDate.setHours(myDate.getHours()+1);
myDay2 = myDate.getDay();
if(myDay1 != myDay2)
{
document.write("In "+myEnum+" hours time it will be tomorrow.<BR>");
}
else
{
document.write("In "+myEnum+" hours time it will still be today.<BR>");
}
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getHours()`, `Date.prototype`, `Date.setUTCHours()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.30

ECMA 262 edition 3 – section – 15.9.5.34

Date.setMilliseconds() (Method)

Sets the milliseconds value of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Time value`	
JavaScript syntax:	-	*myDate*`.setMilliseconds(`*aMillisecondValue*`)`
Argument list:	*aMillisecondValue*	a value between 0 and 999 milliseconds

The computations are done based on extracting a local time value and changing the millisecond count for that, before storing it back again as a local time value.

The result of this method is the new time value having had the milliseconds component set according to the passed-in argument.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDate.setMilliseconds(0);
myTime = myDate.getTime();
document.write("The time rounded down to the nearest second is "  + myTime + "
Milliseconds");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getMilliseconds()`,`Date.prototype`, `Date.setUTCMilliseconds()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.24

ECMA 262 edition 3 – section – 15.9.5.28

Date.setMinutes() (Method)

Sets the minutes of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Time value`
JavaScript syntax:	- `myDate.setMinutes(aMinutesValue)` - `myDate.setMinutes(aMinutesValue, aSecondsValue)` - `myDate.setMinutes(aMinutesValue, aSecondsValue, aMillisecondsValue)`
Argument list:	`aMillisecondsValue` — An optional value between 0 and 999 milliseconds `aMinutesValue` — A value between 0 and 59 minutes `aSecondsValue` — An optional value between 0 and 59 seconds

The computation is done according to local time coordinates.

ECMA mandates that the seconds and milliseconds should be optional arguments. If the milliseconds value is omitted, the current milliseconds value is used as if the value had been supplied with a `getMilliseconds()` method. Likewise, the seconds value behaves as if it had been supplied by a `getSeconds()` method, if it is missing.

The result of this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDate.setMilliseconds(0);
myDate.setSeconds(0);
myDate.setMinutes(0);
myTime = myDate.getTime();
document.write("The time rounded down to the nearest hour is "  + myTime + "
Milliseconds");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getMinutes()`,`Date.prototype`, `Date.setUTCMinutes()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.28

ECMA 262 edition 3 – section – 15.9.5.32

Date.setMonth() (Method)

Sets the month number of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Time value`	
JavaScript syntax:	-	*myDate*`.setMonth(`*aMonthValue*`)`
	- (JavaScript 1.3 +)	*myDate*`.setMonth(`*aMonthValue*`,` *aDateValue*`)`
Argument list:	*aDateValue*	An optional value in days of the month
	aMonthValue	A value between 0 and 11 in months

The result returned by this method will be the new time value of the containing object having been adjusted by the values passed in as arguments.

The computations are carried out in local time coordinates.

ECMA mandates that the date should be an optional argument. If the date value is omitted, the value is provided internally as if the `getDate()` method had been called to provide it.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
myDate  = new Date();
myDate.setDate(4);
myDate.setMonth(6);
document.write("The fourth of July this year is a " + myArray[myDate.getDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getMonth(),Date.prototype,Date.setUTCMonth()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.34

ECMA 262 edition 3 – section – 15.9.5.38

Date.setSeconds() (Method)

Sets the seconds value of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Time value`	
JavaScript syntax:	-	*`myDate`*`.setSeconds(`*`aSeconds Value`*`)`
	- (JavaScript 1.3 +)	*`myDate`*`.setSeconds(`*`aSeconds Value, aMillisecondsValue`*`)`
Argument list:	*`aMillisecondsValue`*	An optional value between 0 and 999 milliseconds
	`aSecondsValue`	A value between 0 and 59 seconds

The values are computed as local time coordinates.

ECMA mandates that the milliseconds value should be an optional argument. If the milliseconds value is omitted, the current milliseconds value is used as if the value had been supplied with a `getMilliseconds()` method.

The result of this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDate.setMilliseconds(0);
myDate.setSeconds(0);
myTime = myDate.getTime();
document.write("The time rounded down to the nearest minute is "  + myTime + "
Milliseconds");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getSeconds()`,`Date.prototype`, `Date.setUTCSeconds()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.26

ECMA 262 edition 3 – section – 15.9.5.30

Date.setTime() (Method)

Sets the time value of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Time value`	
JavaScript syntax:	-	*myDate*`.setTime(`*aTimeValue*`)`
Argument list:	*aTimeValue*	A value measured in milliseconds since midnight on any date.

The time value is extracted by clipping off any overflow past 24 hours. This is done with the internal `TimeClip()` method. The value of the receiving object is set to the resulting value. That same value is returned as the result of the method.

The result is a time clipped to modulo 24 hours.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY onMouseMove="moved()">
Move mouse horizontally to change day number
<DIV ID="ONE" STYLE="background-color:YELLOW">?</DIV>
<SCRIPT>
myArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
myDate = new Date();
function moved(anEvent)
{
myDate.setTime(event.x * 1000000);
document.all.ONE.innerText = myArray[myDate.getDay()];
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getTime(),Date.prototype,TimeClip()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.23

ECMA 262 edition 3 – section – 15.9.5.27

Date.setUTCDate() (Method)

Sets the UTC day within a month of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Time value`	
JavaScript syntax:	-	*`myDate`*`.setUTCDate(`*`aDateValue`*`)`
Argument list:	*`aDateValue`*	A day number within the current month

The result returned by this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

The calculations are performed according to UTC time coordinates.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
myDate = new Date();
myDate.setUTCDate(1);
document.write("The first day of this month is a " + myArray[myDate.getDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getUTCDate()`,`Date.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.33

ECMA 262 edition 3 – section – 15.9.5.37

Date.setUTCFullYear() (Method)

Sets the UTC full year value of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Time value`
JavaScript syntax:	- `myDate.setUTCFullYear(`*`aYearValue`*`)` - *`myDate`*`.setUTCFullYear(`*`aYearValue, aMonthValue`*`)` - *`myDate`*`.setUTCFullYear(`*`aYearValue, aMonthValue, aDateValue`*`)`
Argument list:	*`aDateValue`* — An optional date within the month value *`aMonthValue`* — An optional 0 to 11 month value *`aYearValue`* — A full year value

The result returned by this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

The computations are carried out in UTC time coordinates.

ECMA mandates that the month and date should be optional arguments. If the date value is omitted, the value is provided internally as if the `getUTCDate()` method had been called to provide it. Likewise the month value will be replaced by internally calling the `getUTCMonth()` method.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
myDate = new Date();
myDate.setUTCFullYear(myDate.getFullYear() - 1);
document.write("A year ago on today's date, it was a " +
myArray[myDate.getDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getFullYear()`,`Date.getUTCFullYear()`, `Date.prototype`,`Date.setFullYear()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.37

ECMA 262 edition 3 – section – 15.9.5.41

Date.setUTCHours() (Method)

Sets the UTC hours of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Time value`
JavaScript syntax:	- *myDate*.setUTCHours(*anHoursValue*) - *myDate*.setUTCHours(*anHoursValue*, *aMinutesValue*) - *myDate*.setUTCHours(*anHoursValue*, *aMinutesValue*, *aSecondsValue*) - *myDate*.setUTCHours(*anHoursValue*, *aMinutesValue*, *aSecondsValue*, *aMillisecondsValue*)

Argument list:		
	aMillisecondsValue	An optional value between 0 and 999 milliseconds
	aMinutesValue	An optional value between 0 and 59 minutes
	anHoursValue	A value between 0 and 23 hours
	aSecondsValue	An optional value between 0 and 59 seconds

The value is computed according to UTC time coordinates.

ECMA mandates that the minutes, seconds and milliseconds should be optional arguments. If the milliseconds value is omitted, the current milliseconds value is used as if the value had been supplied with a `getUTCMilliseconds()` method. Likewise, the seconds value behaves as if it had been supplied by a `getUTCSeconds()` method, and the minutes value as if `getUTCMinutes()` was used, if they are missing.

The result of this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDay1 = myDate.getDay();
for(myEnum=1; myEnum<24; myEnum++)
{
myDate.setUTCHours(myDate.getHours()+1);
myDay2 = myDate.getDay();
if(myDay1 != myDay2)
{
document.write("In "+myEnum+" hours time it will be tomorrow.<BR>");
}
else
{
document.write("In "+myEnum+" hours time it will still be today.<BR>");
}
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getHours()`,`Date.getUTCHours()`,`Date.prototype`, `Date.setHours()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.31

ECMA 262 edition 3 – section – 15.9.5.35

Date.setUTCMilliseconds() (Method)

Sets the UTC milliseconds value of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`Time value`

JavaScript syntax:	-	*myDate*.setUTCMilliseconds(*aMillisecondValue*)
Argument list:	*aMillisecondValue*	a value between 0 and 999 milliseconds

The computations are done based on extracting a UTC time value and changing the millisecond count for that before storing it back, again as a UTC time value.

The result of this method is the new time value having had the milliseconds component set according to the passed-in argument.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDate.setUTCMilliseconds(0);
myTime = myDate.getTime();
document.write("The time rounded down to the nearest second is "  + myTime + "
Milliseconds");
</SCRIPT>
</BODY>
</HTML>
```

See also:	Date.getMilliseconds(),Date.getUTCMilliseconds(), Date.prototype,Date.setMilliseconds()

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.25

ECMA 262 edition 3 – section – 15.9.5.29

Date.setUTCMinutes() (Method)

Sets the UTC minutes of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	Time value
JavaScript syntax:	- *myDate*.setUTCMinutes(*aMinutesValue*)
	- *myDate*.setUTCMinutes(*aMinutesValue*, *aSecondsValue*)
	- *myDate*.setUTCMinutes(*aMinutesValue*, *aSecondsValue*, *aMillisecondsValue*)

Argument list:	*aMillisecondsValue*	An optional value between 0 and 999 milliseconds
	aMinutesValue	A value between 0 and 59 minutes
	aSecondsValue	An optional value between 0 and 59 seconds

The computation is done according to UTC time coordinates.

ECMA mandates that the seconds and milliseconds should be optional arguments. If the milliseconds value is omitted, the current milliseconds value is used as if the value had been supplied with a `getUTCMilliseconds()` method. Likewise, the seconds value behaves as if it had been supplied by a `getUTCSeconds()` method, if it is missing.

The result of this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDate.setUTCMilliseconds(0);
myDate.setUTCSeconds(0);
myDate.setUTCMinutes(0);
myTime = myDate.getTime();
document.write("The time rounded down to the nearest hour is "  + myTime + "
Milliseconds");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getMinutes()`, `Date.getUTCMinutes()`, `Date.prototype`, `Date.setMinutes()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.29

ECMA 262 edition 3 – section – 15.9.5.33

Date.setUTCMonth() (Method)

Sets the UTC month number of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0

Property/method value type:	Time value	
JavaScript syntax:	-	*myDate*.setUTCMonth(*aMonthValue*)
	-	*myDate*.setUTCMonth(*aMonthValue, aDateValue*)
Argument list:	*aDateValue*	An optional value in days of the month
	aMonthValue	A value between 0 and 11 in months

The result returned by this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

The computations are carried out in UTC time coordinates.

ECMA mandates that the date should be an optional argument. If the date value is omitted, the value is provided internally as if the `getUTCDate()` method had been called to provide it.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myArray = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
myDate  = new Date();
myDate.setUTCDate(4);
myDate.setUTCMonth(6);
document.write("The fourth of July this year is a " + myArray[myDate.getDay()]);
</SCRIPT>
</BODY>
</HTML>
```

See also:	Date.getMonth(),Date.getUTCMonth(),Date.prototype, Date.setMonth()

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.35

ECMA 262 edition 3 – section – 15.9.5.39

Date.setUTCSeconds() (Method)

Sets the UTC seconds value of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0

Property/method value type:	`Time value`	
JavaScript syntax:	- *myDate*.setUTCSeconds(*aSecondsValue*)	
	- *myDate*.setUTCSeconds(*aSecondsValue*, *aMillisecondsValue*)	
Argument list:	*aMillisecondsValue*	An optional value between 0 and 999 milliseconds
	aSecondsValue	A value between 0 and 59 seconds

The values are computed as UTC time coordinates.

ECMA mandates that the milliseconds value should be an optional argument. If the milliseconds value is omitted, the current milliseconds value is used as if the value had been supplied with a `getUTCMilliseconds()` method.

The result of this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myDate = new Date();
myDate.setUTCMilliseconds(0);
myDate.setUTCSeconds(0);
myTime = myDate.getTime();
document.write("The time rounded down to the nearest minute is "  + myTime + "
Milliseconds");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date.getSeconds()`,`Date.getUTCSeconds()`, `Date.prototype`,`Date.setSeconds()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.27

ECMA 262 edition 3 – section – 15.9.5.31

Date.setYear() (Method)

Sets a non-Y2K compliant year number of the `time` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0 Deprecated in JavaScript 1.3
Property/method value type:	`Time value`
JavaScript syntax:	- *myDate*.setYear(*aYearNumber*)
Argument list:	*aYearNumber* — If the year number is less than 100, then 1900 is added to it.

The result returned by this method is the new time value of the containing object having been adjusted by the values passed in as arguments.

The computation is carried out in local time coordinates.

This method has difficulties with properly resolving the century, which is why it is deprecated and should be replaced by `Date.setFullYear()` in your scripts.

As of ECMA edition 3 it is no longer included in the standard, although implementations may still provide it for backwards compatibility.

Warnings:

- Although it is described in the ECMA standard, it is noted that this function is not formally part of the standard and an ECMA compliant implementation need not provide it. Instead, the `Date.setFullYear()` function should be used.
- Nevertheless, some implementations may support this function although it is strongly recommended that you avoid its use.

See also:	`Date.getFullYear(),Date.getYear(),Date.prototype, Date.setFullYear()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.38

ECMA 262 edition 3 – section – B.2.5

Date.toDateString() (Method)

The value of the `Date` object is presented just as a date.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myDate.toDateString()`*

The time value is rounded off and only the date value is presented by this method.

Cross-references:

ECMA 262 edition 3 – section – 15.9.5.3

Date.toGMTString() (Method)

Converts a `Date` object to a string containing a GMT time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0 Deprecated in JavaScript 1.3	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myDate.toGMTString()`*

This method returns a string value containing a GMT representation of the date and time value in the receiving object. The contents of the string are implementation-dependent, but are intended to represent the date in a convenient, human-readable form. However, this method is now deprecated in favor of `Date.toUTCString()` which should be used in preference. It is provided merely for compatibility with older systems.

As of ECMA edition 3 it is no longer included in the standard although implementations may still provide it for backwards compatibility.

Warnings:

- This function is now deprecated and you should use to`UTCString()` instead.

See also:	`Cast` operator, Cookie expires, `Date.prototype`, `Date.toUTCString()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.41

ECMA 262 edition 3 – section – B.2.6

Date.toLocaleDateString() (Method)

The value of the `Date` object is presented just as a date taking the present locale into consideration.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myDate*`.toLocaleDateString()`

The time value is rounded off, and only the date value is presented by this method. Any locale specific character encoding is performed as necessary. If the implementation is particularly well endowed with internationalization code, this is an opportunity to reorganize the order of the day, month and year components and also to substitute leading zeros if necessary. The month names may also be spelled differently.

Cross-references:

ECMA 262 edition 3 – section – 15.9.5.6

Date.toLocaleString() (Method)

Converts a `Date` object to a string with the locale specific time.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myDate*`.toLocateString()`

A string representing the date taking the locale into account.

The contents of the string that is returned, is implementation dependent. This method is intended to represent the date in a convenient and human-readable form that is appropriate to the geographic or cultural locale that is defined for the hosting environment.

In the Macintosh environment, the format should conform to that set by the date/time control panel in accordance with the Apple guidelines for application developers.

See also:	`Cast` operator, `Date.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.39

ECMA 262 edition 3 – section – 15.9.5.5

Date.toLocaleTimeString() (Method)

The value of the `Date` object is presented just as a time taking the present locale into consideration.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myDate.toLocaleTimeString()`

The date value is removed and only the time value is presented by this method. Any locale specific character encoding is performed as necessary. If the implementation is particularly well endowed with internationalization code, this is an opportunity to substitute leading zeros if necessary.

Cross-references:

ECMA 262 edition 3 – section – 15.9.5.7

Date.toSource() (Method)

Outputs a date formatted as a `Date` literal contained in a string.

Availability:	ECMAScript edition – 3 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myDate`*`.toSource()`

This is an alternative way to deliver a string version of a date value. In this case, it is formatted as a `Date` literal and can then be used in an `eval()` function to assign another date.

If you run the example below (in N 4, or 6), it should yield something resembling this:

```
(new Date(961700949597))
```

Note that the date value is in milliseconds UTC time, and will vary, but you should see a very long number like that.

Example code:

```
// Create a date object and display its source
myObject = new Date();
document.write(myObject.toSource());
```

See also:	`Date.prototype`, `Date.toString()`

Date.toString() (Method)

Return a string primitive version of an object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myDate*`.toString()`

This method returns a string primitive representation of the date value of the receiving object. The contents of the string are implementation-dependent, but are intended to represent the date in a convenient, human-readable form in the current time zone. This is likely to be a UTC time value but implementations may perform some localization.

Warnings:

- If you plan to process the date string, be aware that MSIE presents its date format differently to the Netscape browser. Here is the MSIE presentation style:
  ```
  Wed Jun 21 20:20:03 UTC 2000
  ```
- And this is how Netscape presents the same time value:
  ```
  Wed Jun 21 20:20:03 GMT-2300 (2000
  ```
- Note that one uses UTC time and the other uses GMT. There is a very odd time-zone offset with the Netscape example and it is also missing a closing parenthesis. This may be platform- and version-dependent.
- You can override this behavior by creating your own `toString()` method and associating it with the `Date.prototype`. The example demonstrates how to override `toString()` with your own custom handler that should yield the same result on all platforms.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that overrides toString()
function myToString()
{
myYear  = this.getFullYear();
myMonth = this.getMonth() + 1;
myDate  = this.getDate();

return (myDate + "-" + myMonth + "-" + myYear);
}
// Register the new function
Date.prototype.toString = myToString;
// Create a date and test the replacement Date.toString() method
myDate = new Date();
document.write(myDate.toString())
document.write("<BR>")
</SCRIPT>
</BODY>
</HTML>
```

See also:	Cast operator, `Date.prototype`, `Date.toSource()`, `toString()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.2

ECMA 262 edition 3 – section – 15.9.5.2

Date.toTimeString() (Method)

The value of the `Date` object is presented just as a time.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- `myDate.toTimeString()`

The date value is removed and only the time value is presented by this method.

Cross-references:

ECMA 262 edition 3 – section – 15.9.5.4

Date.toUTCString() (Method)

Converts a `Date` object to a string with UTC time.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myDate*`.toUTCString()`

This method returns a string value. The contents of the string are implementation-dependent, but are intended to represent the date in a convenient, human-readable form in UTC coordinates.

See also:	`Cast` operator, `Date.prototype`, `Date.toGMTString()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.40

ECMA 262 edition 3 – section – 15.9.5.42

Date.UTC() (Method/static)

A class based factory method for converting numeric values to `Date` objects.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`UTC time in milliseconds`	
JavaScript syntax:	-	`Date.UTC(`*aYear, aMonth*`)`
	-	`Date.UTC(`*aYear, aMonth, aDate*`)`
	-	`Date.UTC(`*aYear, aMonth, aDate, anHour*`)`
	-	`Date.UTC(`*aYear, aMonth, aDate, anHour, aMinute*`)`
	-	`Date.UTC(`*aYear, aMonth, aDate, anHour, aMinute, aSecond*`)`
	- (JavaScript 1.3 +)	`Date.UTC(`*aYear, aMonth, aDate, anHour, aMinute, aSecond, aMillisecond*`)`
	aDate	An optional date within the month value

Argument list:	*aMillisecond*	An optional value between 0 and 999 milliseconds
	aMinute	An optional value between 0 and 59 minutes
	aMonth	An optional 0 to 11 month value
	anHour	A value between 0 and 23 hours
	aSecond	An optional value between 0 and 59 seconds
	aYear	A full year value
	aValue	A time in UTC milliseconds

This method returns a date value with the indicated date and time value.

When the `Date.UTC()` method is called, it interprets the arguments as UTC time values and returns a number representing the UTC time in milliseconds.

The value stored in the new date object depends on the argument values that are supplied. The arguments are all optional but are positional so, if an argument is missing, it is assumed to be the last argument and so on.

Functionally, the algorithm that manufactures a new date value uses the internal `MakeDay()`, `MakeTime()` and `MakeDate()` methods that we describe elsewhere.

If the year value is less than 99, then the date creation adds 1900 to it and assumes the date is in the 20th century. To avoid millennium problems, always specify a full year number.

Where arguments are omitted, zero values are assumed for hours, minutes and seconds. When all three are missing, the time is assumed to be midnight.

An incomplete date with zero, one or two arguments will behave in an implementation dependent way and might yield a different result depending on the platform supporting the script interpreter.

See also:	`Date(),Date.constructor,MakeDate(),MakeDay(), MakeTime(),TimeClip()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.4.3

ECMA 262 edition 2 – section – 15.9.4.4

ECMA 262 edition 2 – section – 15.9.4.5

ECMA 262 edition 2 – section – 15.9.4.6

ECMA 262 edition 2 – section – 15.9.4.7

ECMA 262 edition 2 – section – 15.9.4.8

ECMA 262 edition 2 – section – 15.9.4.9

ECMA 262 edition 2 – section – 15.9.4.10

ECMA 262 edition 3 – section – 15.9.4.3

Date.valueOf() (Method)

Returns a number that is the date and time value for the receiving `Date` object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDate*`.valueOf()`

This method returns a numeric value representing the date and time in milliseconds UTC.

See also:	Cast operator, `Date.prototype.valueOf()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.5.3

ECMA 262 edition 3 – section – 15.9.5.8

Day from year (Time calculation)

A date and time algorithm defined by ECMAScript.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

We need to calculate a reference point for each year number to know when it starts. We do this by calculating the number of days since the timer started.

All non-leap years have 365 days with the usual number of days in each month. Leap years have an extra day in February. The calculation shown below uses known leap years and non -eap years to adjust the day numbers, and yield the day number of the first day of the given year:

```
DayFromYear(y) =
365 * (y - 1970) +
floor((y - 1969) / 4) -
floor((y - 1901) / 100) +
floor((y - 1601) / 400)
```

This function will return the start day for the year number passed in as an argument. See the example code to see how to run this in a test harness.

Example code:

```
<HTML>
<BODY>
<SCRIPT>
// Test day from year
document.write("<TABLE BORDER=1>");
for(var ii=1980; ii<2009; ii++)
{
   document.write("<TR>");
   document.write("<TD>");
   document.write(ii);
   document.write("</TD>");
   document.write("<TD>");
   document.write(dayFromYear(ii));
   document.write("</TD>");
   document.write("</TR>");
}
document.write("</TABLE>");

// Day from year function
function dayFromYear(aYear)
{
   var myDay = 365 * (aYear - 1970)  +
      Math.floor((aYear - 1969) / 4)   -
      Math.floor((aYear - 1901) / 100) +
      Math.floor((aYear - 1601) / 400);
   return myDay;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Broken down time, Date from time, Date number, Day within year, Days in year, In leap year, Month from time, Time from year, Time range, Year from time

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.3

ECMA 262 edition 3 – section – 15.9.1.3

Day number (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

Calculating day numbers uses the `Math.floor()` function. It is the largest integer remaining after the time value is divided by the number of milliseconds per day.

The formula for calculating day number is shown here:

t = an instant in time measured in milliseconds relative to 01-January-1970 UTC.

```
msPerDay = 86400000

Day(t) = Math.floor(t/msPerDay)
```

Example code:

```
<HTML>
<BODY>
<SCRIPT>
// Measure time in milliseconds since 01-01-1970 and work
// out the day number
var myDate = Number(new Date());
document.write(dayNumber(myDate));
// Work out day number from milliseconds
function dayNumber(aMillisecondTime)
{
var msPerDay = 86400000
var myDay = Math.floor(aMillisecondTime/msPerDay);

return myDay;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Broken down time, Date from time, Date number, Day within year, Month from time, Month number, Time from year, Time range, Time within day, Week day

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.2

ECMA 262 edition 3 – section – 15.9.1.2

Day within year (Time calculation)

A date and time algorithm defined by ECMAScript.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

The day number within a year is calculated by subtracting the day at the start of the year from the target day being evaluated. The difference is the day number within its year.

The formula for calculating day number is shown here:

t = an instant in time measured in milliseconds relative to 01-January-1970 UTC.

```
msPerDay = 86400000

Day(t) = floor(t/msPerDay)
```

All non-leap years have 365 days with the usual number of days in each month. Leap years have an extra day in February. The calculation shown below uses known leap years and non-leap years to adjust the day numbers and yield the day number of the first day of the given year and then use that to work out the time in milliseconds when the year started:

```
DayFromYear(y) =

365 * (y - 1970) +

floor((y - 1969) / 4) -

floor((y - 1901) / 100) +

floor((y - 1601) / 400)

TimeFromYear(y) = msPerDay * DayFromYear(y)

YearFromTime(t) = The largest integer y to make TimeFromYear(y) less than or equal
to t.

DayWithinYear(t) = Day(t) - DayFromYear(YearFromTime(t))
```

Example code:

```
<HTML>
<BODY>
<SCRIPT>
// Work out a day number within the year
msPerDay = 86400000;
myMilliseconds = Number(new Date());
document.write(dayWithinYear(myMilliseconds));
// Work out day number within year based on time value
function dayWithinYear(aMilliseconds)
{
myDayNumber     = dayNumber(aMilliseconds);
myYearFromTime  = yearFromTime(aMilliseconds);
myDayFromYear   = dayFromYear(myYearFromTime);
myDayWithinYear = myDayNumber - myDayFromYear;

return myDayWithinYear;
}
// Return year number based on time value
function yearFromTime(aMilliseconds)
{
myStartYear = 1970;

while(timeFromYear(myStartYear) < myMilliseconds)
{
myStartYear++
}
```

```
return myStartYear-1;
}
// Work out milliseconds at start of year
function timeFromYear(aYear)
{
myTime = msPerDay * dayFromYear(aYear);
return myTime;
}
// Work out day number from milliseconds
function dayNumber(aMillisecondTime)
{
myDay = Math.floor(aMillisecondTime/msPerDay);

return myDay;
}
// Day from year function
function dayFromYear(aYear)
{
myDay = 365 * (aYear - 1970)             +
Math.floor((aYear - 1969) / 4)   -
Math.floor((aYear - 1901) / 100) +
Math.floor((aYear - 1601) / 400);
return myDay;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Broken down time, Date from time, Date number, Day from year, Day number, Month from time, Month number, Time from year, Year from time

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.4

ECMA 262 edition 3 – section – 15.9.1.4

Daylight savings time adjustment (Definition)

An adjustment to the local time value.

Availability:	ECMAScript edition – 2

Daylight savings time is the practice of shifting the settings of your clock during the Summer or Winter months to move some daylight time from the morning to the evening. The nature of this shift is dependent on locality and cultural issues. The time may be shifted in the Summer relative to standard time or it may be shifted in the Winter. In the UK, during the Winter, time is shifted backwards by an hour. In the USA, clocks are shifted forwards in the Summer.

An ECMA compliant implementation is expected to be able to determine the correct daylight savings time algorithm.

The algorithm is used to determine the Daylight Savings Time Adjustment value (known internally as DaylightSavingTA). This value is measured in milliseconds and must depend only on the following factors:

- The time since the beginning of the year
- Whether the year is a leap year
- The week day of the beginning of the year
- The geographic location

The ECMAScript implementation should decide whether daylight saving time would be in effect according to the currently available algorithm. Historically, daylight saving time algorithms may be different according to local political decisions and it would be too complicated trying to take that into account.

It is possible that the underlying host environment provides some daylight savings time algorithm support and the standard allows that this can be used if it is available.

The standard does not specify any particular algorithm. It just mandates that the algorithm should only rely on a restricted sub-set of information.

Warnings:

- The standard suggests that daylight savings time support is really a matter for the host environment and should be considered implementation and platform dependent. There are portability implications in that respect.

See also:	Broken down time, Calendar time, Date and time, Local time, Local time zone adjustment, Universal coordinated time

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.8

ECMA 262 edition 3 – section – 15.9.1.8

Days in year (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

The number of days in a year depends on whether it is a leap year or not.

ECMA compliant implementations use an extended Gregorian system to map a day number to a year number and to determine the month and date within that year. In this system, leap years are summarized thus:

```
DaysInYear(y) =

365 if ((y modulo 4) != 0)

366 if ((y modulo 4) == 0) and ((y modulo 100) != 0)

365 if ((y modulo 4) == 0) and ((y modulo 400) != 0)

366 if ((y modulo 400) == 0)
```

Example code:

```
<HTML>
<BODY>
<SCRIPT>
// Test number of days in year
document.write("<TABLE BORDER=1>");
for(var ii=1980; ii<2009; ii++)
{
document.write("<TR>");
document.write("<TD>");
document.write(ii);
document.write("</TD>");
document.write("<TD>");
document.write(daysInYear(ii));
document.write("</TD>");
document.write("</TR>");
}
document.write("</TABLE>");
// Day from year function
function daysInYear(aYear)
{
if((aYear % 4) != 0)
{
return 365;
}

if(((aYear % 100) != 0) ||
((aYear % 400) == 0))
{
return 366;
}

return 365;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Broken down time, Day from year, In leap year, Time range, Year number

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.3

ECMA 262 edition 3 – section – 15.9.1.3

DbPool object (Object/NES)

An object of the class "`DbPool`" which provides a means of pooling connections to multiple databases.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
JavaScript syntax:	NES	`myDbPool = new DbPool();`
	NES	`myDbPool = new DbPool(aType, aServer, aUser, aPassword, aDb);`
	NES	`myDbPool = new DbPool(aType, aServer, aUser, aPassword, aDb, maxCon);`
	NES	`myDbPool = new DbPool(aType, aServer, aUser, aPassword, aDb, maxCon, aFlag);`
Argument list:	*aType*	A database type
	aServer	A data source name or server name
	aUser	A user identifier
	aPassword	A valid password for the user
	aDb	A database name if required by the data source
	maxCon	A number indicating maximum connections
	aFlag	Commit/rollback flag value
Object properties:	`prototype`	
Object methods:	`connect(), connected(), connection(), disconnect(), majorErrorCode(), majorErrorMessage(), minorErrorCode(), minorErrorMessage(), storedProcArgs(), toString()`	

In JavaScript 1.2, you can connect to multiple databases and reuse connections. Each database can have a pool of available and ready connections which can be created when needed.

A `DbPool` object is created in a similar way to when you connect a single `database` object to a database. In this case however, you create a new `DbPool` object each time so you could maintain connections to several databases which is not possible with a plain (JavaScript 1.1) `database` object.

To create an object instance of the `DbPool` class, use the new operator on the `DbPool()` constructor.

The database type would likely be one of:

- ORACLE
- SYBASE
- INFORMIX
- DB2
- ODBC

The server value would identify one of the available data source names.

The user and password details would correspond to valid users you have already created on your database server.

Likewise the database name, although for some databases such as Oracle this may be done through the data source description and so the argument should be left blank in that case.

The maximum number of connections depends on your licensing arrangements with your database supplier and the capacity of your server and whether it needs to share connections with other services.

The commit flag indicates whether to commit (`true`) or rollback (`false`) a pending transaction.

See also:	`database.connect()`, Netscape Enterprise Server, `unwatch()`, `watch()`

Property	JavaScript	JScript	NES	Notes
`prototype`	1.2 +	-	3.0 +	-

Method	JavaScript	JScript	NES	Notes
`connect()`	1.2 +	-	3.0 +	-
`connected()`	1.2 +	-	3.0 +	-
`connection()`	1.2 +	-	3.0 +	-
`disconnect()`	1.2 +	-	3.0 +	-
`majorErrorCode()`	1.2 +	-	3.0 +	-
`majorErrorMessage()`	1.2 +	-	3.0 +	-
`minorErrorCode()`	1.2 +	-	3.0 +	-
`minorErrorMessage()`	1.2 +	-	3.0 +	-
`storedProcArgs()`	1.2 +	-	3.0 +	-
`toString()`	1.2 +	-	3.0 +	-

DbPool() (Constructor)

Used for creating new pools of connections to a database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`DbPool` object	
JavaScript syntax:	NES	`new DbPool(`*`aType, aServer, aUser, aPassword, aDb, maxCon, aFlag`*`);`
Argument list:	*`aDb`*	The name of a database
	`aFlag`	Commit or roll back on close
	`aPassword`	A valid password for the user
	`aServer`	The name of a database server
	`aType`	A valid connection type
	`aUser`	A user registered for database access on the server
	`maxCon`	The maximum number of simultaneous connections

When creating a new `DbPool` object, you need to indicate the type of database you are connecting to. The following are examples of commonly available database types:

- ORACLE
- SYBASE
- INFORMIX
- DB2
- ODBC

Use one of these values in the first argument to this method.

You will also need to know the name of your target server, a valid username and password and, if multiple databases are supported by your database server, then you will need to know the name of the target database you want to connect to.

The last two arguments indicate the maximum number of connections available at once and a flag to indicate the commit policy on closure. You can elect to automatically commit any changes (dangerous) or roll back any uncommitted changes.

See also:	`database.connect()`

DbPool.connect() (Method)

Connects the `DbPool` object to a database.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
JavaScript syntax:	NES	`myDbPool.connect(`*`aType, aServer, aUser, aPassword, aDb, maxCon, aFlag`*`);`
Argument list:	*aDb*	The name of a database
	aFlag	Commit or rollback on close
	aPassword	A valid password for the user
	aServer	The name of a database server
	aType	A valid connection type
	aUser	A user registered for database access on the server
	maxCon	The maximum number of simultaneous connections

When connecting to a database, you need to indicate the type of database you are connecting to. The following are examples of commonly available database types:

- ORACLE
- SYBASE
- INFORMIX
- DB2
- ODBC

Use one of these values in the first argument to this method.

You will also need to know the name of your target server, a valid username and password and, if multiple databases are supported by your database server, then you will need to know the name of the target database you want to connect to.

The last two arguments indicate the maximum number of connections available at once and a flag to indicate the commit policy on closure. You can elect to automatically commit any changes (dangerous) or roll back any uncommitted changes.

See also:	`database.connect()`

DbPool.connected() (Method)

A flag that indicates the connection status for this `DbPool` object.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	`myDbPool.connected()`

This method returns a Boolean value that tells you whether the `database` object is connected to a database or not.

See also:	`database.connected()`

DbPool.connection() (Method)

Requests a connection object from the pool of available connections.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`Connection object`	
JavaScript syntax:	NES	`myDbPool.connection(`*`aName, aTimeout`*`)`
Argument list:	*`aName`*	A connection name
	`aTimeout`	Timeout in seconds

The object returned by this method is used to maintain the connection state details between the Netscape Enterprise Server and the back end database it is retrieving data from.

See also:	`Connection` object

DbPool.disconnect() (Method)

Disconnect from the database discarding all connections in the pool in the process.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
JavaScript syntax:	NES	`myDbPool.disconnect()`

It is a good idea to disconnect from the database when you know you won't need it anymore. This is good practice and allows other processes to connect when resources are scarce.

Until the database is connected again, only the `connect()` and `connected()` methods have any meaning.

See also:	`database.disconnect()`

DbPool.majorErrorCode() (Method)

Returns an error code value for an error that happened in the database or the interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myDbPool`*`.majorErrorCode()`

For a status code value of 5 when using the Oracle database, this yields a return code from the Oracle Call-level interface.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the SQL server message number.

For a status code value of 7 when using the Informix database, this yields the Informix error code.

For a status code value of 7 when using the Sybase database, this yields the DB-Library error number.

See also:	`Connection.majorErrorCode()`, `database.majorErrorCode()`, Error handling

DbPool.majorErrorMessage() (Method)

Returns an error message text for an error that happened in the database or the interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myDbPool`*`.majorErrorMessage()`

For a status code value of 5 when using the Oracle database, this yields a text string describing the server error.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields a text string from SQL server.

For a status code value of 7 when using the Informix database, this yields the text string from the vendor error library.

For a status code value of 7 when using the Sybase database, this yields a text string from the DB-Library.

See also:	`Connection.majorErrorMessage()`, `database.majorErrorMessage()`, Error handling

DbPool.minorErrorCode() (Method)

Returns a supplementary error code value for an error that happened in the database or the interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myDbPool`*`.minorErrorCode()`

For a status code value of 5 when using the Oracle database, this yields an operating system error code from the Oracle Call-level interface.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the severity level from SQL server.

For a status code value of 7 when using the Informix database, this yields the ISAM error code.

For a status code value of 7 when using the Sybase database, this yields the severity level of the error from the DB-Library.

See also:	`Connection.minorErrorCode()`, `database.minorErrorCode()`, Error handling

DbPool.minorErrorMessage() (Method)

Returns a supplementary error message text for an error that happened in the database or the interface to it.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myDbPool`*`.minorErrorMessage()`

For a status code value of 5 when using the Oracle database, this yields the Oracle server name.

For a status code value of 5 when using SQL server through the ODBC database interface, this yields the SQL server name.

For a status code value of 7 when using the Informix database, this yields a text string describing the ISAM error.

For a status code value of 7 when using the Sybase database, this yields the text of the operating system error from the DB-Library.

See also:	`Connection.minorErrorMessage()`, `database.minorErrorMessage()`, Error handling

DbPool.prototype (Property)

The prototype for the `DbPool` object that can be used to extend the interface for all `DbPool` objects.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`DbPool` object	
JavaScript syntax:	NES	`DbPool.prototype`
	NES	*`myDbPool`*`.constructor.prototype`

Refer to:

prototype property

DbPool.storedProcArgs() (Method)

Creates a prototype for a stored procedure and controls the argument passing.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`Stproc` object	
JavaScript syntax:	NES	*`myDbPool`*`.storedProcArgs()`

The prototype stored `procedure` object supports input and output parameters. This prototype object is used to indicate the direction of values in these parameters.

See also:	`database.storedProcArgs()`

DbPool.toString() (Method)

Returns a string primitive version of the `DbPool` object.

Availability:	JavaScript – 1.2 Netscape Enterprise Server – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`myDbPool.toString()`

The value of the object is converted to a string value that represents its value.

See also:	`database.toString()`

DD object (Object/HTML)

An object that represents the `<DD>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myDD* = *myDocument*.all.*anElementID*
	IE	*myDD* = *myDocument*.all.*tags("DD")*[*anIndex*]
	IE	*myDD* = *myDocument.all*[*aName*]
	-	*myDD* = *myDocument*.getElementById(*anElementID*)
	-	*myDD* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myDD* = *myDocument*.getElementsByTagName ("*DD*")[*anIndex*]
HTML syntax:	`<DD> ... </DD>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object

Object properties:	`noWrap`
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`

This object represents the definition value of an item in a definition list. It corresponds to a related definition term maintained in an `DT` object. `DT` and `DD` objects are paired up and maintained together as a member of the `DL` collection.

The `<DD>` tag is a block-level tag. That means that it forces a line break before and after itself unless the `DL` is compacted.

See also:	`DL` object, `DT` object, `Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`noWrap`	-	3.0 +	-	4.0 +	-	-	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

DD.noWrap (Property)

A switch to control text wrapping within the <DD> block.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myDD*.noWrap

This is a Boolean value that controls whether the textual content is wrapped at the right hand window border or not.

If the value `false` is assigned to this property, then words will wrap as the page is drawn. This is good and is the way you would expect a browser to behave. The text will flow according to the space available.

If the value `true` is assigned to this property, the line of text will continue to the right until a <BR> or other block level tag is encountered. This will force the horizontal width of the page to extremely large and the user will need to scroll furiously to be able to see the text and then scroll back again for the start of the next line.

Warnings:

- Only use this if you plan to place line breaks at frequent intervals yourself and really do need to control the line breaks manually.

Debugger (Definition)

A tool to help with the location of bugs in your script.

See also:	Constraint, Debugging – client side, Error handling, JavaScript debugger console

debugger (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Debugging - client side (Advice)

How to debug faulty client-side scripts.

When you are trying to debug client-side code, there are not many ways you have available for seeing what is going on. Usually, you have to resort to placing a `document.writes()` into the body of the page or perhaps putting up an alert box when the script runs.

In Netscape, you can bring up a debugging console by typing the string '`javascript:`' into the location box and pressing `return`. Because you have not indicated any action to be passed to the interpreter, it opens the debugging window by default.

Another way to do some very effective debugging is to trap the `onError` event handler with your own debugger script. You would implement the debugging error handler as something that opens a window and places HTML into it. Then you attach the handler to the current window with the `self.onerror` property.

You might find it useful to open a document window with a "`text/plain`" MIME type so you can simply use `document.writeln()` actions to display some debugging text.

A useful technique when debugging form handlers is to loop through all the elements in the form, presenting them in an `alert()` box for inspection. Of course this is useful for all kinds of collections, not just forms.

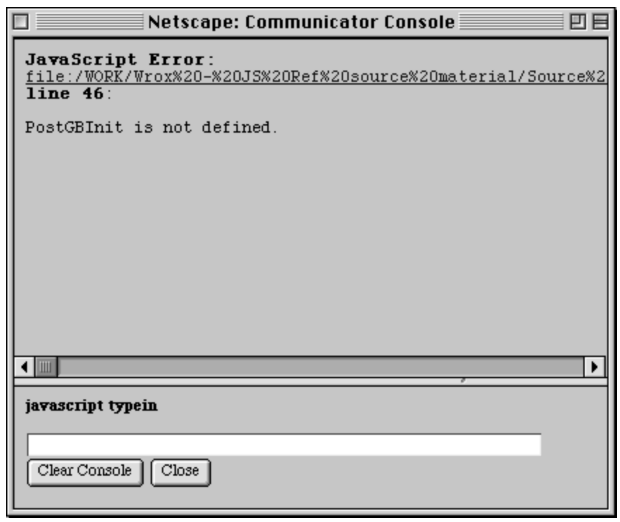
The example script illustrates three different ways to do debugging. There are lots of other techniques too.

The first example simply uses `document.write()` and `alert()` methods to feed information back to the user.

The second example is more sophisticated and attaches an error handler to the `onError` event hook. This works in MSIE. In Netscape, it was expected to work, but as of the pre-release version PR3 it still would not catch the errors. Since Netscape is very likely to undergo rapid and frequent updates for the foreseeable future, this might start to work quite soon. Nevertheless, Netscape provides the JavaScript console which may be helpful too.

The third example is based around a call to the embedded Java VM and requests that it prints a message on its console display surface. Again, whether this works may be browser and platform dependent but it is a starting point for custom building your own suite of debugging tools.

The fourth example shows how to debug form values without causing a form submit or page refresh.



Example code:

```
<!-- Simple debugging example -->
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
// Write text into the document body at this point
document.write('Your text here');
// Present a dialog box with a message
alert('Your other text here');
</SCRIPT>
</BODY>
</HTML>
-----------------------------------------------------------------
<!-- Custom debugging example -->
<HTML>
<HEAD>
<SCRIPT>
function debugHandler(aMessage, aURL, aLineNumber)
{
   myText  = "An error has occurred\n\nThe message is\n\n";
   myText += aMessage + "\n\n at URL\n\n";
   myText += aURL + "\n\non line number ";
   myText += aLineNumber;

   alert(myText);

   return true;
}
</SCRIPT>
</HEAD>

<BODY>
<SCRIPT>
// Note that this may not work in Netscape Navigator
self.onerror = debugHandler("This is an example error", "local site", 21);
myError = null.open();
</SCRIPT>
</BODY>
</HTML>
-----------------------------------------------------------------
// Java console debugging
// Calls the Java console output to print a debug message
function consoleDebug(errMessage)
{
   java.lang.System.out.println(errMessage);
}

someError = "true";

// Call the console debug output like this
if(someError == true)
{
   consoleDebug("Display this text on the Java Console");
}
```

```
-----------------------------------------------------------------
// A form debugger provided by Jon Stephens
// This can pasted into the location window if it is prefixed
// with javascript: and if all the line breaks and extra
// spaces are removed
function formDebug()
{
   var output = "";
   for (var i = 0; i < document.forms.length; i++)
   {
      for (var j = 0; j < document.forms[i].elements.length; j++)
      {
         output +=  "Form: " + i
            +  " Elem: " + j
            +  " Type: "
            +  document.forms[i].elements[j].type  + "; "
            +  document.forms[i].elements[j].name  + ": "
            +  document.forms[i].elements[j].value + "<br>"
      }
   }
   var newWin = window.open("","newWin","width=350,height=350");
   with(newWin.document)
   {
      open();
      write(output);
      close();
   }
}

formDebug();
```

See also: `Arguments.callee`, `Arguments.caller`, Debugging – server side, Dialog boxes, `Document.open()`, Input-output, JavaScript debugger console, Object inspector, `Window.alert()`, `Window.confirm()`, `Window.onerror`, `Window.prompt()`

Cross-references:

Wrox *Instant JavaScript* – page – 205

Web-references:

http://msdn.microsoft.com/scripting/default.htm?/scripting/debugger/
http://developer.netscape.com/tech/javascript/index.html?content=/software/jsdebug.html

Debugging - server side (Definition)

How to debug faulty stand-alone scripts.

Debugging stand-alone, and server-side scripts is essentially the same.

You don't necessarily have a `document.write()` method but you will certainly have a means of printing text. In the case of a server-side script, that may be the way you construct the response so you can put debugging information in the response, so that it can be viewed by from the web browser.

You might not want to do that if you are debugging some security related services though. In that case, your web server probably has a logging capability and you might be able to write to the error log.

Example code:

```
// Nombas ScriptEase debugging is like thisClib.puts('Your message text here');
```

See also:	Debugging – client side, Input-output

Decimal point (.) (Delimiter)

A delimiter that marks the beginning of the fractional part of a floating point value.

Availability:	ECMAScript edition – 2 Opera – 3.0

A decimal point separates the integer and fractional parts of a floating point value. The character being used may need to alter depending on the locale of the hosting environment.

If your national language routines change the formatting of numbers, be aware that commas and dots mean different things.

A dot is really a decimal point. A comma is a thousands separator.

In some formatting regimes, thousands may be separated by a space character and a comma may be used in place of a dot (or vice versa).

Note also that it is a convention in financial reports to show negative values in parentheses but as a positive value.

All of this can make it difficult to parse numeric values that a user may enter into a text field.

Warnings:

- Localization of JavaScript implementations is still undergoing some development.

See also:	Floating point, Localization, Object property delimiter (.), Property accessor

Cross-references:

ECMA 262 edition 2 – section – 7.7.3

ECMA 262 edition 3 – section – 7.8.3

Decimal value (Definition)

A numeric value based on a radix of 10.

A decimal value is an integer composed of only the following characters:

```
0 1 2 3 4 5 6 7 8 9
```

Decimal values are never prefixed by a zero character.

The sequence carries over for the next increment when each column reaches the value 9. Thus:

```
0 1 2 3 4 5 6 7 8 9 10 11 12
```

Converting between decimal, hexadecimal, and binary values is quite complicated. Here are the first 16 values in all number bases:

Binary	Octal	Decimal	Hexadecimal
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

Warnings:

- Beware when you prefix decimal values with a zero character. You may want to justify a column of figures. If you add leading zero characters to a numeric string, if that string is subsequently parsed back to a numeric value, you may inadvertently export the value as a decimal but import it as an octal value. This can lead to an extremely difficult to diagnose fault in your software because the parsers sometimes add some intelligence and will correctly interpret the value as decimal if the characters 8 or 9 are present but otherwise interpret it as octal notation.
- Be careful that you remove any leading zero characters from the text strings that you plan to convert using the numeric parser.
- This may be implementation dependent behavior to some extent.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Create a 0-255 lookup table
document.write("<TABLE BORDER=1>");
document.write("<TR><TH>Index</TH>");
document.write("<TH>Binary</TH>");
document.write("<TH>Octal</TH>");
document.write("<TH>Hex</TH>");
document.write("<TH>Char</TH></TR>");
for(ii=0; ii<256; ii++)
{
```

```
    myBinary        = ii.toString(2);
    myBinaryPadding = "00000000".substr(1,(8-myBinary.length));

    myOctal         = ii.toString(8);
    myOctalPadding  = "0000".substr(1,(4-myOctal.length));

    myHex           = ii.toString(16);
    myHexPadding    = "00".substr(1,(2-myHex.length));

    myChar          = String.fromCharCode(ii);

    document.write("<TR ALIGN=RIGHT><TD>");
    document.write(ii);
    document.write("</TD><TD>");
    document.write(myBinaryPadding+myBinary);
    document.write("</TD><TD>");
    document.write(myOctalPadding+myOctal);
    document.write("</TD><TD>");
    document.write(myHexPadding+myHex.toUpperCase());
    document.write("</TD><TD>");
    document.write(" "+myChar);
    document.write("</TD></TR>");
}
document.write("</TABLE>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	Hexadecimal value, Integer constant, `Number.toString()`, Octal value

Declaration (Definition)

Declares the attributes of an identifier.

A declaration gives the type and scope of an identifier.

It specifies whether the identifier refers to a function or a variable and indicates what its name is. The scope is indicated by the placement. Variables are scoped globally if they are created outside of a function script source block.

When storage is being allocated, a declaration is sometimes called a definition. This means you might declare a function and define a variable. If you indicate the size of an array then that would be a definition.

See also:	Definition, Initialization, JavaScript language, Scope chain, Variable Declaration

Declared function (Definition)

A function can be declared in the script source text.

Availability:	ECMAScript edition – 2

Declared functions are added to the script source text to provide additional functions that are only needed while that script is being executed.

Declaring a function in a script requires a function declaration, which includes the name of the function, its arguments and a block of source script code to be executed when the function is called.

Through the Prototype Inheritance mechanism, this script code will be called in preference to any identically named function higher up the inheritance chain. This provides a way to override methods in parent object classes, which mimics (but is not identical to) the sub-class/super-class relationships in class-based languages.

See also:	Function object, function(...) ...

Cross-references:

ECMA 262 edition 2 – section – 10.1.1

ECMA 262 edition 3 – section – 10.1.1

decodeURI() (Function)

This ECMA defined function can be used to decode an entire URI value that was encoded with the `encodeURI()` function.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`decodeURI(`*`anEncodedURI`*`)`
Argument list:	*`anEncodedURI`*	A previously encoded URI

This function is the complement of the `encodeURI()` function that we describe elsewhere.

A string that might have been encoded with the `encodeURI()` function either locally or remotely, can be converted back to a normal URI string with this function.

As far as ECMAScript is concerned, this supersedes the `unescape()` function, which is flagged as deprecated functionality in the JScript 5.5 documentation.

Note that the hash character (`#`) is not decoded.

See also:	`decodeURIComponent()`, `encodeURI()`, `encodeURIComponent()`, `escape()`, `unescape()`, URI handling functions

Cross-references:

ECMA 262 edition 3 – section – 15.1.3.1

decodeURIComponent() (Function)

This ECMA defined function can be used to decode a URI component value that was encoded with the `encodeURIComponent()` function.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`decodeURIComponent(`*`anEncodedComponent`*`)`
Argument list:	*`anEncodedComponent`*	A previously encoded component

This function is the complement of the `encodeURIComponent()` function that we describe elsewhere.

A string that might have been encoded with the `encodeURIComponent()` function either locally or remotely, can be converted back to a normal URI string with this function.

This enhances the encode/decode facilities of the compliant browser over and above what used to be available with `escape()/unescape()` functions.

See also:	`decodeURI(),encodeURI(),encodeURIComponent(), escape(),unescape()`, URI handling functions

Cross-references:

ECMA 262 edition 3 – section – 15.1.3.2

Decrement value (–) (Operator/postfix)

Pre or post decrementing operator.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`--anOperand`*
	-	*`anOperand--`*
Argument list:	*`anOperand`*	A numeric value that can be decremented

The operand is decremented by one.

A pre-fixing decrementor will subtract 1 from the operand value before it is used in the expression.

A post-fixing decrementor will subtract 1 from the operand after it is used in the expression.

Be careful how you use this pre/post placement as you can easily generate 'off by one' errors in your algorithms by placing the operator on the wrong side of the operand.

This operator is more or less functionally equivalent to:

```
anOperand -= 1
```

which is equivalent to:

```
anOperand = anOperand - 1
```

See also:	Additive expression, Additive operator, Increment value (++), Negation operator (-), Postfix decrement (--), Postfix expression, Postfix increment (++), Prefix decrement (--), Prefix expression, Prefix increment (++)

Cross-references:

ECMA 262 edition 2 – section – 11.3.2

ECMA 262 edition 3 – section – 11.3.2

Deep copying (Definition)

Making a duplicate of objects, property by property.

See also:	`Array.toSource()`, Copying objects

default: (Label)

A target label for use with the switch statement as a catch-all for any unmatched cases.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server – 3.0

See also:	Label, Selection statement, switch(...) ... case: ... default: ...

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

ECMA 262 edition 3 – section – 12.11

defaultStatus (Property)

The default status text of the window.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`defaultStatus`
	-	`defaultStatus = aString`
	-	*`myWindow`*`.defaultStatus`
	-	*`myWindow`*`.defaultStatus = aString`
Argument list:	*`aString`*	A new value for the default status

Refer to:

Window.defaultStatus

DefaultValue() (Function/internal)

Internal private function.

Availability:	ECMAScript edition – 2

This internal function returns the default value for the object, given that the caller indicates a preferred result type in the hint argument.

The hint gives some suggestion to the receiving object about the preferred result type. This is a fairly ambiguous result. Generally if you ask for a string you will get a string. Asking for a number will generally yield a number. However, there are cases where the `DefaultValue` for an object cannot be rendered into the preferred type and you may generate a run-time error.

If you don't specify any hint value at all, the `DefaultValue` will first assume you want a Number unless the receiver is a `Date` object in which case it assumes a String is required.

See also:	Internal Method

Cross-references:

ECMA 262 edition 2 – section – 8.6.2.6

ECMA 262 edition 3 – section – 8.6.2.6

Defensive coding (Advice)

Allowing your scripts to be downwards compatible and coding for portability and robustness.

Coding defensively is to check for the existence of a feature before using it. Also to check that objects are defined before trying to modify their properties. You can check the version of the browser and switch various features of your scripts on and off accordingly.

With a little thought and planning, you can contrive your script design so that it degrades gracefully if it is run on less capable browser versions than that which you originally designed it for.

Defensive coding also covers cros- platform and cross-browser support. Running scripts in different browsers often exposes the differences between them. This may be simply a property value that is available in one context but not another. A more serious problem might be an entire object class that is missing from particular implementations.

Sometimes the differences are more subtle than that, and you may just find that a document object model needs to be traversed in a different way, depending on the browser being used.

Example code:

```
// Check for the existence of an object before using it
if(document.layers)
{
   document.write(document.layers);
}
else
{
   document.write("There is no layers array in this browser!");
}
```

See also:	Browser version compatibility, Compatibility, Cross browser compatibility

Definition (Definition)

Defines the storage for an identifier.

A definition is a declaration that also allocates storage for the item being declared.

For example:

```
new Array();
```

creates an empty `Array` object. As there is no indication of the likely size the array will require, no storage is allocated for array elements.

Now consider this:

```
new Array(10);

new Array("aaa", "bbb", "ccc");
```

Each of these declarations require some storage to be allocated. In the first entry, space for 10 elements is reserved in the array. In the second, there are three element pointers created and the storage for the three items as well.

Thus we have a definition.

See also:	Argument, Compliance, Compound statement, Constraint, Declaration, Deprecated functionality, Diagnostic message, `false`, Function, `function( ... ) ...`, Implementation, Object, Parameter, RValue, `true`, Variable Declaration

DEL object (Object/HTML)

An object that represents a `<DEL>` HTML tag within the document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myDEL* = *myDocument*.all.*anElementID*
	IE	*myDEL* = *myDocument*.all.tags("DEL")[*anIndex*]
	IE	*myDEL* = myDocument.all[aName]
	-	myDEL = *myDocument*.getElementById(*anElementID*)
	-	*myDEL* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myDEL* = *myDocument*.getElementsByTagName("DEL")[*anIndex*]
HTML syntax:	`<DEL> ... </DEL>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`cite, dateTime`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

You can add mark-up to a document to strike through text as if it were deleted. This deleted text block can also have a citation reference that links to a description of why the deletion took place.

The DOM level 1 specification includes this in the `ModElement` object functionality.

See also:	`Element` object, `INS` object, `ModElement` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
cite	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
dateTime	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

DEL.cite (Property)

A URL that references a document that describes why the item was deleted.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- `myDEL.cite`

The URL of the document that describes why the text was marked as deleted is noted in this property.

DEL.dateTime (Property)

The date and time that the deletion occurred.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- `myDEL.dateTime`

This is the date and time value for when the deletion change occurred. If you are maintaining change control down to the sub-document level in a content management system, these values can be defined from change records in the database.

delete (Operator/unary)

Property deletion operator.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 3.0 Internet Explorer – 4.0 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	`delete anObject`
	-	`delete myArray[anIndex]`
	-	`delete myObject.aProperty`
Argument list:	*anIndex*	The index of the item to be deleted from the array.
	anObject	An object to be deleted
	aProperty	An object property to be removed

The delete operator is used to delete a property from an object or delete a reference to an object. It can also be used to delete an element from an array.

Using the new operator, a new object is created with a reference count of zero. Assigning that object creation to a variable increments the reference count. Saving it as an object property also increments the reference count. Storing it in an array does likewise.

All of these are simply references to the same object.

Deleting a variable containing a reference to an object decrements the reference count for that object.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

Warnings:

- The delete operator will not work with MSIE version 3.02 or earlier. It also won't work in Netscape 2.02.
- Netscape does not generate an error in version 3.0 if the delete operator is used but it is functionally ignored and does nothing.
- In JavaScript version 1.0 and version 1.1, the delete operator does not actually delete any object properties. Instead it sets them to null. If you then subsequently test for the existence of those properties, they might not prove `false` when you expect them to.
- In the JavaScript 1.2 implementation, in version 4 of Netscape , the behavior of the delete operator as it applies to variables is slightly different. Any variable declared with the `var` statement is considered to be permanent and cannot be deleted. This will generate an error if you try. This happens for the case when you enclose the script in `<SCRIPT LANGUAGE="JAVASCRIPT1.2">` tags.
- There is some difference between Netscape and MSIE as to whether global variables can be deleted. The safest assumption is that they cannot, so to remain portable, your script should not try.

Example code:

```
// Create a new object
myObject = new Object;
// Then delete a reference to it
delete myObject;
```

See also:	Associativity, Garbage collection, Grouping operator ( ), `JSObject.removeMember()`, Memory leak, `Object` object, Operator Precedence, Property attribute, Reference counting, Unary expression, Unary operator, Variable statement

Cross-references:

ECMA 262 edition 2 – section – 11.1.4

ECMA 262 edition 2 – section – 11.4.1

ECMA 262 edition 3 – section – 11.4.1

Wrox *Instant JavaScript* – page – 21

Wrox *Instant JavaScript* – page – 28

Delete() (Function/internal)

Internal private function.

Availability:	ECMAScript edition – 2

This internal function removes the named property from the object.

This function does not walk the prototype inheritance chain. If it did and a shared property got deleted, that property would disappear from ALL the objects that shared it.

If the property does not exist in the receiving object, the `Delete()` is assumed to have been successful anyway and `true` is returned.

If the property is present but has the `DontDelete` attribute, it cannot be removed so `false` is returned.

If the property can be deleted, then it is removed and `true` is returned.

See also:	Internal Method

Cross-references:

ECMA 262 edition 2 – section – 8.6.2.5

ECMA 262 edition 3 – section – 8.6.2.5

Deprecated functionality (Pitfall)

Some language features are to be discontinued in later versions of the language.

Deprecated functionality describes a feature that is likely to be removed and should not be used in new developments.

There are various features that interpreter manufacturers add from time to time and which become standardized in a different and possibly better way, or that everyone agrees should be disregarded. Netscape's data tainting functionality is a good example of a dead end that was introduced and withdrawn soon after.

The following is a list of such deprecated functionality that should be avoided:

- Data tainting
- JavaScript extensions to HTML character entities

Warnings:

- If you use deprecated functionality, your script may fail when it is deployed on other platforms or when the platform it is hosted on is revised.

See also:	Definition, JavaScript entity, Obsolescent, Pitfalls

Desktop JavaScript (Definition)

Control of desktop automation with JavaScript.

You can install a JavaScript interpreter that you can use to automate activity on your desktop. On a Macintosh, this kind of activity has been possible for some time with HyperCard, AppleScript and the Macintosh Programmer's Workshop shell environment. Now you can install products such as Nombas ScriptEase.

On Windows, you could have used DOS batch files and if you had installed the NeXTStep developer tools in the past, a Bourne shell was possible. Other alternatives have also been available. Now, you can use JScript within the Windows Script Host environment. You can also use ScriptEase on Windows.

On UNIX, a variety of possibilities are available. ScriptEase again is useful but there are other freely available JavaScript interpreters that you can download and build yourself. It just depends how much work you want to do before you can start scripting.

See also:	Host environment, Platform, Server-side JavaScript

Cross-references:

Wrox *Instant JavaScript* – page – 5

detachEvent() (Method)

A means of detaching events from windows and documents that were previously attached with the `attachEvent()` method.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	`detachEvent(`*`anEventName`*`)`
	IE	*`myWindow`*`.detachEvent(`*`anEventName`*`)`
Argument list:	*`anEventName`*	The name of an event to be handled

See also:	`.htc`, `<STYLE>`, `Document.attachEvent()`, `Document.detachEvent()`, `Window.attachEvent()`, Window.detachEvent()

Determining the object type (Useful tip)

To determine what kind of object type you have, this function may be useful.

This checks the type of the value passed in. If it is an object, then it looks at the constructor and returns the constructor's name. This works in Netscape 4.7 and MSIE version 5 and should work on other browsers, but check before deploying that it works on all the browsers you need it to.

Example code:

```
<HTML>
<BODY>
<SCRIPT>
// Determine object type
function what_kind(aValue)
{
var myType = typeof(aValue);
if (myType == "object")
{
return aValue.constructor.name;
}
else
{
return myType;
}
}
document.write(what_kind(1));
</SCRIPT>
</BODY>
</HTML>
```

See also:	`constructor.name`

Developing JavaScript source code (Definition)

Techniques for easing developer headaches.

Refer to:

Preferences

DFN object (Object/HTML)

An object representing the HTML content delimited by the `<DFN>` HTML tags.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myDFN* = *myDocument*.all.*anElementID*
	IE	*myDFN* = *myDocument*.all.tags("DFN")[*anIndex*]
	IE	*myDFN* = *myDocument*.all[*aName*]
	-	*myDFN* = *myDocument*.getElementById(*anElementID*)

JavaScript syntax:	-	*myDFN* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myDFN* = *myDocument*.getElementsByTagName("DFN")[*anIndex*]
HTML syntax:	<DFN> ... </DFN>	
Argument list:	*anElementID*	The ID value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

Refer to:

Element object

DHTML (Standard)

Dynamic HTML controlled by JavaScript. A fourth generation browser technology for dynamically altering the document that describes a web page.

You might be forgiven for thinking DHTML is a new kind of HTML. Actually it's not really. It's just plain old HTML code. What makes it dynamic are the supporting technologies that operate on it. Without them, there is nothing dynamic about HTML at all.

Principally, it's the JavaScript and CSS style sheet support that makes HTML dynamic and it's the access to the HTML through a Document Object Model that allows us to operate on the HTML in the document.

Things are complicated a little because the dynamism can be applied at the server-end and really what we mean by DHTML is a client-end activity. Server created HTML, even though it's done with JavaScript, is not really DHTML.

Perhaps a less ambiguous description is CSS and DOM scripting. That allows some differentiation of the appearance vs. structural things you can accomplish with script driven documents.

See also:	Dynamic HTML

DHTML Behavior (Definition)

A mechanism for enhancing the dynamic capabilities of HTML.

Refer to:

`Element.addBehavior()`

Diagnostic message (Definition)

A message from the interpreter warning you about a script error.

A diagnostic message is what the implementation presents when it detects an error.

These are normally presented as an `alert` box in web browsers.

In other implementations such as Web Server back ends, server-side JavaScript failures may be recorded in an error log or written into the pages being sent to the client. However, you should be careful to remove any debugging stubs before launching your service as these messages could potentially divulge some valuable security related information about your server (and how to hack it).

You should always know where the error logs are written for your implementation. On a UNIX platform, it is helpful to open an additional terminal console and pipe the output of the error log to the screen. This can be done with the tail command like this:

```
tail -f error.log
```

See also:	Constraint, Definition, Error, Error handling

Dialog boxes (Definition)

User communication is effected by means of several different dialog boxes.

Dialog boxes are generally modal. That means they need to be acted on and dismissed by the user. Some are scriptable and others are not. Your operating system will determine whether modality extends across the whole system or just to the application. If modality is system-wide, you won't be able to switch applications while a modal dialog is on display.

Here are a few that you might encounter routinely when scripting:

- `Window.alert()`
- `Window.confirm()`
- `Window.prompt()`
- `Window.showHelp()`
- `Window.showModalDialog()`
- `Window.showModelessDialog()`

One that you cannot script is the secure login panel presented to allow you to authenticate your access to a secure web page. That is controlled by the server, and your browser should not provide any script-driven interface to it.

See also:	Debugging – client side, `Window.alert()`,`Window.confirm()`, `Window.prompt()`,`Window.showHelp()`, `Window.showModalDialog()`,`Window.showModelessDialog()`

Dialog object (Object/JScript)

This is the `parent` object of a frame within a modal dialog window.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Inherits from:	`Window` object
JavaScript syntax:	IE *myDialog* = *myFrame*.parent IE *myDialog* = *myWindow*.parent
Event handlers:	`onAfterPrint, onBeforePrint, onBeforeUnload, onBlur, onDragDrop, onError, onFocus, onHelp, onLoad, onMouseMove, onMove, onResize, onScroll, onUnload`

To all intents and purposes this is a `window` object except that, because it is modal, it is placed on top of all other windows and, until it is disposed of, no other browser windows can be accessed.

Some early documentation suggests that there are dialog sizing properties for this window type but this has not been confirmed to work.

On the new version 6.0 Netscape browser, there is the tantalizing possibility that the appearance of dialog boxes can be affected by altering the 'skin' of the browser. As yet, this does not appear to be scriptable or accessible from HTML. It is highly likely that internally the appearance is JavaScript driven. Historically, much of the internal behavior of Netscape has been controlled by `.js` files so this may change.

See also:	`Window.alert()`,`Window.confirm()`,`Window.parent`, `Window.prompt()`,`Window.showHelp()`, `Window.showModalDialog()`,`Window.showModelessDialog()`

Event name	JavaScript	JScript	N	IE	Opera	Notes
onAfterPrint	-	5.0 +	-	5.0 +	-	-
onBeforePrint	-	5.0 +	-	5.0 +	-	-
onBeforeUnload	-	3.0 +	-	4.0 +	-	-
onBlur	-	3.0 +	-	4.0 +	-	Warning
onDragDrop	-	-	-	-	-	-
onError	-	3.0 +	-	4.0 +	-	Warning
onFocus	-	3.0 +	-	4.0 +	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	Warning
onLoad	-	3.0 +	-	4.0 +	-	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	Warning
onMove	-	-	-	-	-	-
onResize	-	3.0 +	-	4.0 +	-	Warning
onScroll	-	3.0 +	-	4.0 +	-	-
onUnload	-	3.0 +	-	4.0 +	-	Warning

Inheritance chain:

`Window` object

dialogArguments (Property)

The arguments passed to a modal dialog window.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`dialogArguments`
	IE	`myWindow.dialogArguments`

Property attributes:

`ReadOnly`

Refer to:

`Window.dialogArguments`

dialogHeight (Property)

The height of a modal or modeless dialog window.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`dialogHeight`
	IE	*`myWindow`*`.dialogHeight`

Refer to:

`Window.dialogHeight`

dialogLeft (Property)

The offset to the left edge of a modal or modeless dialog window. In IE 4, the default measure is the `em`, in IE 5 it is the `pixel`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`dialogLeft`
	IE	*`myWindow`*`.dialogLeft`

Refer to:

`Window.dialogLeft`

dialogTop (Property)

The offset to the top edge of a modal or modeless dialog window. In IE 4, the default measure is the `em`, in IE 5 it is the `pixel`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`dialogTop`
	IE	*`myWindow`*`.dialogTop`

Refer to:

`Window.dialogTop`

dialogWidth (Property)

The width of a modal or modeless dialog window. In IE 4, the default measure is the `em`, in IE 5 it is the `pixel`.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Property/method value type:	Number primitive	
JavaScript syntax:	IE	dialogWidth
	IE	myWindow.dialogWidth

Refer to:

Window.dialogWidth

Dictionary object (Object/JScript)

A name-value collection object created by the Active X facilities.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myDictionary* = new ActiveXObject("Scripting.Dictionary");
Object properties:	Count	
Object methods:	Add(), Exists(), Item(), Items(), Key(), Keys(), Remove(), RemoveAll()	

A dictionary is a special kind of collection or array which allows you to access items using name and value pairs. This is functionally similar to using associative name indexing on a JavaScript array, so although this is not generally available, its fundamental usefulness is already provided in the native JavaScript implementation.

The Dictionary object provided by JScript and Active X, has a few extra properties and methods to help manage the contents of the dictionary. These could be simulated fairly easily and added to the Array prototype to create your own portable Dictionary-like object.

Warnings:

- As this object needs to be created with the Active X facilities, it is not supported outside of the Windows environment. It is odd that such a useful object can only be constructed in this way and that it does not have a native JavaScript constructor. That at least would make it available in all platforms that JScript runs in.
- Note that the properties and methods for the Dictionary object all start with an upper case letter. This is not the common practice in JavaScript and may cause you a few run-time errors if you forget.
- There are some unusual syntactical constructs in this object. In particular, item and key methods that behave like properties. These are intended to provide an interface to replace an item in the dictionary or to rename a key. These methods would normally be made with several arguments to a single method, but instead they use a method to retrieve a reference to a Dictionary pocket and you can then assign a value to the reference that was returned.

Example code:

```
// Create a new dictionary
var myDictionary = new ActiveXObject("Scripting.Dictionary");
// Store some items in the dictionary
// (Melting Points of fats/waxes)
myDictionary.Add("Butter",       "28");
myDictionary.Add("Lard",         "36");
myDictionary.Add("MuttonTallow", "44");
myDictionary.Add("Beeswax",      "61");
myDictionary.Add("Stearin",      "71.6");
myDictionary.Add("ParaffinWax",  "38");
// Display one item if it exists
if(myDictionary.Exists("ParaffinWax"))
{
document.write("Paraffin Wax melts at ");
document.write(myDictionary.Item("ParaffinWax"));
document.write(" degrees Centigrade.<BR>");
}
// Remove an item
if(myDictionary.Exists("Stearin"))
{
myDictionary.Remove("Stearin");
}
// Change an item and its key
if(myDictionary.Exists("MuttonTallow"))
{
myDictionary.Item("MuttonTallow") = "40";
myDictionary.Key("MuttonTallow") = "BeefTallow";
}
// List all the items
myArray = (new VBArray(myDictionary.Keys())).toArray();
for(myEnum=0; myEnum<myArray.length; myEnum++)
{
document.write("Key value: ");
document.write(myArray[myEnum]);
document.write("  Item value: ");
document.write(myDictionary.Item(myArray[myEnum]));
document.write("<BR>");
}
// Now discard the items in the array
myDictionary.RemoveAll();
```

See also:	ActiveX, `ActiveXObject` object

Property	**JavaScript**	**JScript**	**N**	**IE**	**Opera**	**Notes**
`Count`	-	3.0 +	-	4.0 +	-	ReadOnly.

Method	**JavaScript**	**JScript**	**N**	**IE**	**Opera**	**Notes**
`Add()`	-	3.0 +	-	4.0 +	-	-
`Exists()`	-	3.0 +	-	4.0 +	-	-

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	Notes
`Item()`	-	3.0 +	-	4.0 +	-	Warning
`Items()`	-	3.0 +	-	4.0 +	-	-
`Key()`	-	3.0 +	-	4.0 +	-	-
`Keys()`	-	3.0 +	-	4.0 +	-	Warning
`Remove()`	-	3.0 +	-	4.0 +	-	-
`RemoveAll()`	-	3.0 +	-	4.0 +	-	-

Dictionary.Add() (Method)

Adds a new item to the `Dictionary`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myDictionary*.Add(*aKey, aValue*)
Argument list:	*aKey*	A textual key name
	aValue	A value to store for the key

This method adds a new item to the `Dictionary` associating it with the key name being passed.

Keys are meant to be strings but will be coerced as necessary during the addition. However, you cannot use an `Array` object to build the key name. If you do need to build a key name from an array, you must convert it to a string first.

The items being associated with `Dictionary` keys can be of any type.

You cannot replace an item with this method as a run-time error is caused by an attempt to add a new object where the key name has already been used.

Dictionary.Count (Property)

Returns a count of the number of items in the `Dictionary`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myDictionary*.Count

This behaves like the `length` property of a native `JavaScript` Array or `Collection` object. Note that its name begins with an upper case letter.

Property attributes:

`ReadOnly`

Dictionary.Exists() (Method)

Returns a flag indicating whether a key has an item associated with it in the `Dictionary`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myDictionary`*`.Exists(`*`aKey`*`)`
Argument list:	*`aKey`*	A textual key name

Dictionaries are a useful way to collect items together at random intervals. The `Dictionary` can then be tested later on after the items have been added. This method provides a way to test whether a specific name-value pair has been added to the `Dictionary`.

Dictionary.Item() (Method)

Returns a reference to the `Item` container for a key.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Reference to an item pocket`	
JavaScript syntax:	IE	*`myDictionary`*`.Item(`*`aKey`*`)`
	IE	*`myDictionary`*`.Item(`*`aKey`*`) = `*`aValue`*
Argument list:	*`aKey`*	A textual key name
	`aValue`	A value to store for the key

This method provides a way to extract an item from the `Dictionary` using its key name as an accessor.

This method can also be used as an alternative to the `Dictionary.add()` method although its syntax for doing so is unusual.

These two lines of code are syntactically different but functionally the same when creating a new name-value pair:

```
myDictionary.add("KEY1", "an item text");

myDictionary.item("KEY1") = "an item text";
```

This suggests that you can use the `add()` method to replace an item but in fact that will cause an error. The `item()`, method, used as if it were an `LValue` property is the only way to replace an item content.

Warnings:

- Although this is a method, because it has parentheses and therefore can be called, it also behaves as a property. This somewhat bends the syntax rules for JavaScript and looks like a badly formed expression when you see the variant that does an assignment.
- It is important to use this item within a conditional code block that tests for the existence of the key first. This is because an attempt to retrieve a named item from a `Dictionary` will create a named but empty item if it does not already exist.
- Note the capitalised name of the method.

Dictionary.Items() (Method)

Returns an array containing all Items in the `Dictionary`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`VBArray object`	
JavaScript syntax:	IE	`myDictionary.Items()`

The entire set of items is returned in a `VBArray`. If you want to access this as a normal JavaScript array, you should use the `toArray()` method on the `VBArray` that is created by the `Items()` method.

Dictionary.Key() (Method)

Returns a reference to the named key container.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Reference to a key name pocket`	
JavaScript syntax:	IE	*`myDictionary`*`.Key(`*`anOldKey`*`) = `*`aNewKey`*
Argument list:	*`anOldKey`*	An existing key name
	`aNewKey`	A new name to rename the existing key to

This method provides a reference to the receptacle containing the key name. This 'by-reference' access to the key allows the key to be changed. You would not be able to do that if it accessed the key name by value.

Because you are referencing the container of the key, you can use this method as an `LValue` and assign a new value to the key. This is effectively a key rename mechanism.

If the key does not already exist, like the `Item()` method, this will create a new named but empty item in the `Dictionary`.

Dictionary.Keys() (Method)

Returns a collection of all the keys currently defined in the `Dictionary`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`VBArray object`	
JavaScript syntax:	IE	*`myDictionary`*`.Keys()`

The entire set of key names is returned in a `VBArray`. If you want to access this as a normal JavaScript array, you should use the `toArray()` method on the `VBArray` that is created by the `Items()` method.

Warnings:

- This is not the same as a collection of `Items`. This is a list of key names.

Dictionary.Remove() (Method)

Remove a key name and its item value from the `Dictionary`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*`myDictionary`*`.Remove(`*`aKey`*`)`
Argument list:	*`aKey`*	A textual key name

You should test that the key already exists before attempting to remove it. This will save you generating run-time errors if you attempt to remove items for non-existent keys.

Dictionary.RemoveAll() (Method)

Remove all key names and item values from the `Dictionary`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*`myDictionary`*`.RemoveAll()`

The dictionary is completely emptied and can be re-used instead of creating a new one.

Digit (Definition)

A decimal numeric character.

A digit is any of the following characters:

0 1 2 3 4 5 6 7 8 9

See also:	Identifier, Nondigit

DIR object (Object/HTML)

A somewhat deprecated object that is now superseded by the `<UL>` HTML tag and its object representation. This object represents the contents of a `<DIR>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myDIR* = *myDocument*.all.*anElementID*
	IE	*myDIR* = *myDocument*.all.tags("DIR")[*anIndex*]
	IE	*myDIR* = *myDocument*.all[*aName*]
	-	*myDIR* = *myDocument*.getElementById(*anElementID*)
	-	*myDIR* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myDIR* = *myDocument*.getElementsByTagName("DIR")[*anIndex*]
HTML syntax:	`<DIR> ... </DIR>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`compact`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

The DOM level 1 specification refers to this as a `DirectoryElement` object.

See also:	`Element` object, `UL` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`compact`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

DIR.compact (Property)

An attribute that controls the display of `<DIR>` items and the amount of space they require on the screen.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *myDir*.compact

Refer to:

`DL.compact`

disableExternalCapture() (Method)

Part of the Netscape 4 event propagation complex.

Availability:	JavaScript – 1.2 Netscape – 4.0
Property/method value type:	`undefined`
JavaScript syntax:	N `disableExternalCapture()` N *myWindow*.disableExternalCapture()
See also:	`Window.enableExternalCapture()`, `Window.disableExternalCapture()`

DIV object (Object/HTML)

An object that represents a <DIV> block level element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myDIV* = *myDocument*.all.*anElementID*
	IE	*myDIV* = *myDocument*.all.tags("DIV")[*anIndex*]
	IE	*myDIV* = *myDocument*.all[*aName*]
	-	*myDIV* = *myDocument*.getElementById(*anElementID*)
	-	*myDIV* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myDIV* = *myDocument*.getElementsByTagName("DIV")[*anIndex*]
HTML syntax:	`<DIV> ... </DIV>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align, dataFld, dataFormatAs, dataSrc`	
Event handlers:	`onAfterUpdate, onBeforeUpdate, onBlur, onChange, onClick, onDblClick, onDragStart, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onScroll, onSelectStart`	

A `DIV` object is a means of building a block structured organization of a document that is custom-designed. It is yet another means of building a hierarchical model if none of the existing models fits the structure you need. It is also a way of marking out blocks of a document to be treated as a styled area or perhaps a means of dynamically replacing or modifying document content in an easily controlled manner.

The <DIV> tag is a block-level tag. That means that it forces a line break before and after itself.

The example shows how to exchange the style values between two <DIV> blocks.

Warnings:

- In Netscape 4, a <DIV> object whose CSS style defines its positioning as absolute, is then enumerated in the layers collection at an appropriate position in the document/layer hierarchy. This is confusing because a <DIV> is not a <LAYER>. Refer to the discussion about `Layer` objects in a separate topic for further details.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="ONE" STYLE="background-color:RED">
The DIV block ONE
</DIV>
<DIV ID="TWO" STYLE="background-color:BLUE">
The DIV block TWO
</DIV>
<FORM>
<INPUT TYPE="button" VALUE="CLICK ME" onClick="clickMe()">
</FORM>
<SCRIPT>
function clickMe()
{
myStyle1 = document.all.ONE.style.cssText;
myStyle2 = document.all.TWO.style.cssText;

document.all.ONE.style.cssText = myStyle2;
document.all.TWO.style.cssText = myStyle1;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Element` object, Hierarchy of objects, `Layer` object, `LayerArray` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`dataFld`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`dataFormatAs`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`dataSrc`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
`onChange`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`OnKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onResize`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
`onRowEnter`	-	3.0 +	-	4.0 +	-	-	-	-
`onRowExit`	-	3.0 +	-	4.0 +	-	-	-	-
`onScroll`	-	3.0 +	-	4.0 +	-	-	-	-
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

DIV.align (Property)

The alignment for content within a `<DIV>` block.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	*`myDIV`*`.align`

The alignment of the `DIV` block object with respect to its containing parent object is defined in this property. The expected and widely available set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

Divide (/) (Operator/multiplicative)

Divide one operand by another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`anOperand1 / anOperand2`
Argument list:	*`anOperand1`*	The dividend
	`anOperand2`	The divisor

The left-hand operand is divided by the right-hand operand and the quotient is returned.

The left operand is the dividend, and the right is the divisor. ECMAScript compliant interpreters do not perform integer division. The operand and results of all divisions are double-precision floating point numbers. All divisions are performed according to IEEE 754 specifications.

If either operand is `NaN` then the result is `NaN`.

The sign of the result is positive if both operands have the same sign and negative if the operands have different signs.

Division of an infinity by an infinity results in `NaN`.

Division of an infinity by zero results in an infinity. The sign is determined by the rule stated earlier.

Division of infinity by a non-zero finite value results in a signed infinity. The sign is determined as before.

Division of a finite value by infinity results in zero. The sign is determined as usual.

Division of a zero by a zero results in `NaN`.

Division of zero by any other finite value results in zero with the sign determined as normal.

Otherwise, where there is neither an infinity or zero involved and neither value is `NaN`, the quotient is computed and rounded to the nearest representable value. If the magnitude is too large to represent, it will overflow and become an infinity. If the magnitude is too small to represent, an underflow occurs and the result will be zero.

The associativity is from left to right.

Refer to the operator precedence topic for details of execution order.

See also:	Associativity, Divide then assign (/=), Double-precision, Multiplicative expression, Multiplicative operator, Operator Precedence, Remainder (%)

Cross-references:

ECMA 262 edition 2 – section – 11.5.2

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.5.2

Divide then assign (/=) (Operator/assignment)

Divide one operand by another and put the result in the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`anOperand1 /= anOperand2`*
Argument list:	*`anOperand1`*	The dividend
	`anOperand2`	The divisor

Divide the left operand by the right operand and assign the quotient to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 / anOperand2;
```

Although this is classified as an assignment operator, it is really a compound of an assignment and a multiplicative operator.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

The new value of `anOperand1` is returned as a result of the expression.

Warnings:

- The operand to the left of the operator must be an `LValue`. That is, it should be able to take an assignment and store the value.

See also:	Assignment operator, Associativity, Divide (/), LValue, Multiplicative operator, Operator Precedence, Remainder (%)

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

DL object (Object/HTML)

An object that represents a definition list defined by a `<DL>` HTML tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	`myDL = myDocument.all.anElementID`
	IE	*myDL* = *myDocument*`.all.tags("DL")[`*anIndex*`]`
	IE	*myDL* = *myDocument*`.all[`*aName*`]`
	-	*myDL* = *myDocument*`.getElementById(`*anElementID*`)`
	-	*myDL* = *myDocument*`.getElementsByName(`*aName*`)[`*anIndex*`]`
	-	*myDL* = *myDocument*`.getElementsByTagName("DL")[`*anIndex*`]`
HTML syntax:	`<DL> ... </DL>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`compact`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

Definition lists are collections of member definitions. The whole list is encapsulated in a `DL` object. Each member definition is constructed from a `DT` and a `DD` object which are maintained together.

The `<DL>` tag is a block-level tag. That means that it forces a line break before and after itself.

The DOM level 1 standard describes this as a `DListElement` object.

See also:	`DD` object, `DT` object, `Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`compact`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

DL.compact (Property)

An attribute that controls the display of <DD> and <DT> items and the amount of space they require on the screen.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myDL*.compact

The `DT` and `DD` objects are normally displayed on two consecutive lines. When the `DL` that owns a collection of `DT/DD` object pairs has the `compact` property set to `true`, the `DT/DD` pair are displayed on the same line to save space.

do ... while(...) (Iterator)

A variant of the `while` iterator that checks the condition after execution.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	`aLabel: do { someCode } while (aCondition );`
Argument list:	*aCondition*	This must prove true for a subsequent cycle to start
	aLabel	An optional identifier that names the loop
	someCode	The code that gets executed in the loop

A `do loop` is a variation on the `while` iterator. A `while` iterator checks the condition and only executes the code block if it is `true`. This means that a `while` loop may never execute even once. A `do` iterator checks the condition once the code has been executed. This ensures that a `do` iterator will perform at least one execution of the code block even if the condition proves `false` the first time it is tested.

ECMA edition 2 compliance merely states that it is a reserved word. At edition 3 of the ECMAScript standard, it is a fully supported requirement of compliance.

JavaScript version 1.2 anticipates this and provides it anyway.

If a labelled `continue` is used (available from version 1.2 of JavaScript), it is intended that execution should drop to the bottom of the loop and test the condition again before cycling or falling out.

Note carefully the line that increments the counter. If you leave it out, you create an endless loop and the browser locks you out. Maybe it will eventually crash but you may need to wait a long time.

Warnings:

- In Netscape 4, there is a bug with labelled `continue` statements and `do ... while` loops that causes the `continue` to vector to the top of the loop without testing the condition. This can set up an endless loop. You could work round this by creating a `while` loop and modifying the test condition.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myCounter = 10;
do
{
document.write(myCounter);
```

```
document.write("<BR>");
myCounter++;
}
while(myCounter < 35);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`continue`, Flow control, `for( ... ) ...`, `for( ... in ... ) ...`, Label, Off by one errors, `while( ... ) ...`

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

ECMA 262 edition 3 – section – 12.6.1

Doctype object (Object/DOM)

An object that represents the document type DTD.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Inherits from:	`Node` object	
JavaScript syntax:	IE	*myDoctype* = *myDocument*.all.*anElementID*
	IE	*myDoctype* = *myDocument*.all.tags("DOCTYPE")[*anIndex*]
	IE	*myDoctype* = *myDocument*.all[*aName*]
	-	*myDoctype* = *myDocument*.doctype
	-	*myDoctype* = *myDocument*.getElementById(*anElementID*)
	-	*myDoctype* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myDoctype* = *myDocument*.getElementsByTagName("DOCTYPE")[*anIndex*]
HTML syntax:	`<!DOCTYPE>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`name`	
Collections:	`entities[], notations[]`	

Every document should own a `Doctype` object according to the DOM level 1 specification. This object encapsulates name and some collections that describe the DTD. Work is still underway on standardizing the XML and DTD requirements and this object is therefore likely to change in later versions of the DOM specification.

The DOM level 2 specification adds the following new properties:

- `publicId`
- `systemId`
- `internalSubset`

See also:	`Document.doctype`, `Element.document`, `Notation` object

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`name`	1.5 +	5.5 +	6.0 +	5.5 +	-	1 +	ReadOnly

Inheritance chain:

Node object

Doctype.entities[] (Collection)

A named node map containing all the general entities within the DTD.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`NamedNodeMap object`	
JavaScript syntax:	-	*`myDoctype`*`.entities`

Property attributes:

`ReadOnly`.

Doctype.name (Property)

The name of the DTD that the Doctype encapsulates.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myDoctype.name`

Property attributes:

ReadOnly.

Doctype.notations[] (Collection)

A named node map containing the notations declared in the DTD encapsulated by the `Doctype` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0
Property/method value type:	`NamedNodeMap object`
JavaScript syntax:	- *myDoctype*.notations
See also:	`Notation` object

Property attributes:

`ReadOnly`

Document (Object model)

An organized collection of objects that represent a document.

The `document` object is the foundation around which a scriptable interface to an HTML or XML document is constructed. This is sometimes referred to as the DOM and is subject to its own standardization exercise being managed by W3C and other interested parties.

The DOM has its origins in the MSIE version 4 browsers. Version 3 of MSIE and versions of Netscape prior to version 6 implement a more miscellaneous collection of objects that behave a bit like a DOM but are not really a standards-compliant model.

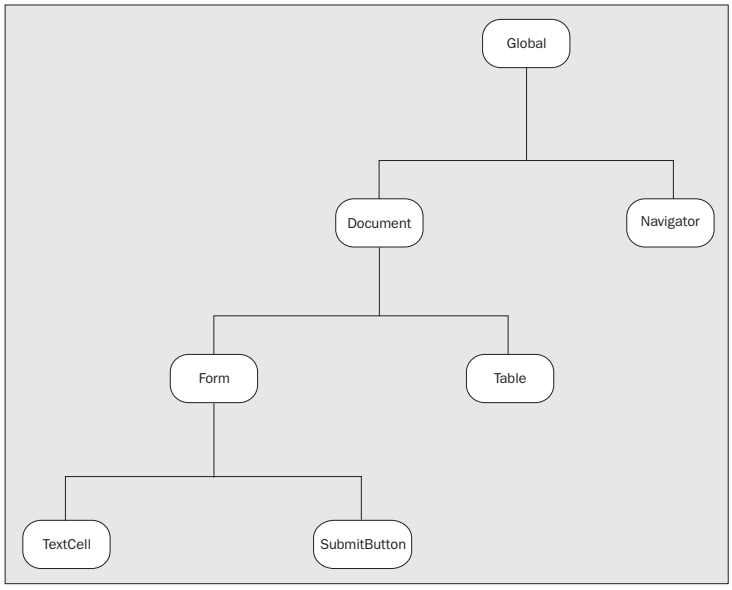
There are areas where the so called DOM support in each browser is so different as to render any script access to the document either problematical or virtually impossible with the same script. This means that you need to support parallel development of HTML and JavaScript to be able to cover both competing browsers.

Now that Netscape 6.0 converges on the same standard- based DOM model as MSIE, we can look forward to a much more portable future for our scripts. So long as we can disregard legacy versions and steer clear of the still quite large number of differences, we should be able to to do much more across different browsers without needing to code differently for each one. History suggests it is also equally likely that they will diverge in other areas where they introduce new features.

The starting point for the DOM hierarchy is the `<HTML>` HTML tag, although the `<BODY>` HTML tag is realistically the root of the DOM.

The basic approach to the DOM differs between the browsers on these points:

- Netscape prior to version 6.0 generally provides a constructor for every object type. MSIE only provides them for objects that you can reasonably instantiate. The new Netscape should comply with the DOM requirements and only provide constructors where they are mandated by the standard.
- MSIE implements an `Element` object on which most other DOM components are based. Netscape 6 implements a structured, DOM compliant model so this is implicit.
- The DOM hierarchy is organized in different ways. MSIE provides many reference vectors for locating parent and child objects to traverse the DOM tree. Netscape provides very few prior to version 6.0.
- MSIE provides an object to represent every tag. Its type is the tag name in upper case.
- On the down-side, MSIE supports a DOM structure that resembles the DOM standard quite closely. However, many of its class names are incorrect. Netscape 6.0 supports a similar structure but uses the correct names for object classes. If you need to use class names in your scripts, beware!



Warnings:

- This is not supported at all on Netscape 2 and 3 or MSIE version 3. You should at least check for these browsers and generate a helpful warning message or skip round the requirement somehow.

See also:	Compatibility, Document object, DOM, Event handler scope

Web-references:

http://msdn.microsoft.com/redirs/inetsdkredir.asphttp://www.w3.org/TR/WD-DOM/

document (Property)

The document within the current window.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0
Property/method value type:	`Document object`
JavaScript syntax:	- `document` - *`myWindow`*`.document`
See also:	`Window.document`, `Document object`

Property attributes:

`ReadOnly`

Document component (Definition)

A component object within a document object model.

Component objects within the document correspond to tags in the HTML or XML document. As tags, they may be given names with the `NAME="..."` HTML tag attribute. In the level 0 DOM, these names become properties of the `document` object, which can lead to pollution of the property/method namespace. This is likely to be changed in DOM level 1.

See also:	DOM

Document event handlers (Definition)

A property containing a reference to an event handler property.

Property/method value type:	Function object

There is a small set of event handler function properties created by default. You can assign your own handlers to these properties (if necessary creating the properties that don't already exist).

Here is a list of all the event handler property names that we discovered during our research:

Event	Handler	Usage
`Abort`	`onabort`	When image loading is aborted.
`AfterPrint`	`onafterprint`	When printing has just finished.
`AfterUpdate`	`onafterupdate`	When an update is completed.
`Back`	`onback`	The user has clicked on the [`BACK`] button in the toolbar.
`BeforeCopy`	`onbeforecopy`	Immediately before a copy to the clipboard.
`BeforeCut`	`onbeforecut`	Immediately before a cut to the clipboard.
`BeforeEditFocus`	`onbeforeeditfocus`	Immediately before the edit focus is directed to an element.
`BeforePaste`	`onbeforepaste`	Immediately before the clipboard is pasted.
`BeforePrint`	`onbeforeprint`	Immediately before printing begins.
`BeforeUnload`	`onbeforeunload`	Called immediately prior to the window being unloaded.
`BeforeUpdate`	`onbeforeupdate`	Called immediately before an update commences.
`Blur`	`onblur`	When an input element loses input focus.
`Bounce`	`onbounce`	Triggered when a marquee element hits the edge of its element area.
`Change`	`onchange`	When edit fields have new values entered or a popup has a new selection, this event's handler can check the new value.
`Click`	`onclick`	When the user clicks the mouse button on the `Element` object that represents the object on screen.
`ContextMenu`	`oncontextmenu`	Special handling for contextual menus.
`Copy`	`oncopy`	When a copy operation happens.
`Cut`	`oncut`	When a cut operation happens.
`DataAvailable`	`ondataavailable`	Some data has arrived asynchronously from an applet or data source.
`DataSetChanged`	`ondatasetchanged`	A data source has changed the content or some initial data is now ready for collection.
`DataSetComplete`	`ondatasetcomplete`	There is no more data to be transmitted from the data source.
`DblClick`	`ondblclick`	When the user double clicks on an object.
`Drag`	`ondrag`	When a drag operation happens.
`DragDrop`	`ondragdrop`	Some data has been dropped onto a window.
`DragEnd`	`ondragend`	When a drag ends.
`DragEnter`	`ondragenter`	When a dragged item enters the element.
`DragLeave`	`ondragleave`	When a dragged item leaves the element.
`DragOver`	`ondragover`	While the dragged item is over the element.

Table continued on following page

Event	Handler	Usage
DragStart	ondragstart	The user has commenced some data selection with a mouse drag.
Drop	ondrop	When a dragged item is dropped.
Error	onerror	Triggered if an error occurs when loading an image.
ErrorUpdate	onerrorupdate	An error has occurred in the transfer of some data from a data source.
FilterChange	onfilterchange	A filter has changed the state of an element or a transition has just been completed.
Finish	onfinish	A `marquee` object has finished looping.
Focus	onfocus	When the form element is selected for entry.
Forward	onforward	The user has clicked on the [`FORWARD`] button in the toolbar.
Help	onhelp	The user has pressed the [`F1`] key or selected [`help`] from the toolbar or menu.
KeyDown	onkeydown	When a key is pressed.
KeyPress	onkeypress	Pressing the key down and releasing it again elicits this event.
KeyUp	onkeyup	When a key is released.
Load	onload	When an object has completed loading.
LoseCapture	onlosecapture	When an element loses event capturing permission.
MouseDown	onmousedown	When the mouse button is pressed.
MouseDrag	onmousedrag	An event handler for mouse drag operations.
MouseMove	onmousemove	When the mouse pointer is moved.
MouseOut	onmouseout	When the mouse pointer leaves the active area occupied by the `Element` object that represents the object on screen.
MouseOver	onmouseover	When the mouse pointer enters the active area owned by the object.
MouseUp	onmouseup	When the mouse button is released.
Move	onmove	The browser window has been moved.
Paste	onpaste	When a paste operation happens.
PropertyChange	onpropertychange	When an object property is modified (similar to the Netscape `watch()` method).
ReadyStateChange	onreadystatechange	An object in the window has changed its ready state.
Reset	onreset	The user has clicked a reset button in a form.
Resize	onresize	As the window is resized, this event is triggered.
RowEnter	onrowenter	The data in a field bound to a data source is about to be changed.

Table continued on following page

Event	Handler	Usage
`RowExit`	`onrowexit`	The data in a field bound to a data source has been changed.
`Scroll`	`onscroll`	The window has been scrolled.
`Select`	`onselect`	Some textual content in the window has been selected.
`SelectStart`	`onselectstart`	A select action is beginning.
`Start`	`onstart`	A marquee element is beginning its loop.
`Stop`	`onstop`	When a stop action occurs.
`Submit`	`onsubmit`	The user has clicked on the submit button in a form.
`Unload`	`onunload`	Triggered when the document is unloaded.

See also:	`Document` object, `Element.onevent`

Document object (Object/HTML)

An object that represents the document currently loaded into the window. This exposes the contents of the HTML document through a variety of collections and properties.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0
Inherits from:	`Element` object

JavaScript syntax:		
	-	`myDocument = document`
	-	*`myDocument`*`= document.documentElement`
	IE	*`myDocument`*` = `*`myElement`*`.document`
	IE	*`myDocument`*` = `*`myElement`*`.offsetParent`
	-	*`myDocument`*` = `*`myElement`*`.ownerDocument`
	IE	*`myDocument`*` = `*`myElement`*`.parentElement`
	-	*`myDocument`*` = `*`myElement`*`.parentNode`
	-	*`myDocument`*` = `*`myFrame`*`.document`
	N	*`myDocument`*` = `*`myLayer`*`.document`
	-	*`myDocument`*` = `*`myWindow`*`.document`
	-	*`myDocument`*` = opener.document`
	-	*`myDocument`*` = self.document`

Argument list:	*`anIndex`*	An index that selects this document

Object properties:	<form_name>, activeElement, alinkColor, background, bgColor, body, characterset, charset, cookie, defaultCharset, designMode, doctype, documentElement, domain, expando, fgColor, fileCreatedDate, fileModifiedDate, fileSize, height, implementation, lastModified, linkColor, location, parentWindow, protocol, readyState, referrer, selection, title, uniqueID, URL, vlinkColor, width
Object methods:	attachEvent(), captureEvents(), clear(), close(), contextual(), createAttribute(), createCDATASection(), createComment(), createDocumentFragment(), createElement(), createEntityReference(), createProcessingInstruction(), createStyleSheet(), createTextNode(), detachEvent(), elementFromPoint(), execCommand(), getElementById(), getElementsByName(), getElementsByTagName(), getSelection(), handleEvent(), mergeAttrbutes(), open(), queryCommandEnabled(), queryCommandIndeterm(), queryCommandState(), queryCommandSupported(), queryCommandText(), queryCommandValue(), recalc(), releaseEvents(), routeEvent(), write(), writeln()
Functions:	captureEvents(), releaseEvents(), routeEvent()
Event handlers:	onAfterUpdate, onBeforeCut, onBeforEditFocus, onBeforePaste, onBeforeUpdate, onClick, onContextMenu, onCut, onDblClick, onDrag, onDragEnd, onDragEnter, onDragLeave, onDragOver, onDragStart, onDrop, onErrorUpdate, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onPaste, onPropertyChange, onReadyStateChange, onRowEnter, onRowExit, onSelectStart, onStop
Collections:	all[], anchors[], applets[], classes[], embeds[], forms[], frames[], ids[], images[], layers[], links[], plugins[], scripts[], styleSheets[], tags[]

The `document` object is the root of a hierarchy that describes the document in terms of objects, properties and methods that can operate on those objects.

The DOM level specification describes a core `Document` object and distinguishes that from the HTMLDocument that is a sub-class of it. However, the browsers do not make such a fine distinction and so Documents and HTMLDocuments are considered to be one and the same. This works because an HTMLDocument inherits the behavior of a DOM core `Document` object.

Documents and their child objects can respond to events by means of event handler functions. These are generally associated with one another by means of the HTML tag attributes that correspond to each event.

Although there is a superset of all event types, each object type only responds to a few of them.

The `document` object is based on the `Element` object, therefore it inherits all the properties and methods of that class and adds others itself.

The `document` object is basically derived initially from the `<BODY>` HTML tag, although it contains some properties that are associated with the `<HEAD>` HTML tag and others from the `<HTML>` HTML tag that encloses the entire file. The document type header also affects properties in the `document` object.

Traversing the document object model in MSIE is quite straightforward. In Netscape prior to version 6.0 it is so difficult as to be virtually impossible. You can access certain parts of the DOM in earlier Netscape browsers by virtue of the forms, applets, embeds and other collections but you cannot access other parts of the DOM at all.

Event handling support via properties containing function objects was added to `Anchor` objects at version 1.1 of JavaScript and is significantly extended in Netscape 6.0 where it supports DOM level 2 capabilities.

Because you might refer to documents in many ways, possibly by means of object properties or as a property belonging to another window, it is not safe to assume that the document property belonging to the `Global` object is always the `document` object you are trying to access. Indeed, a document may belong to a window, frame, layer or Iframe and several may be accessible at once. Because of this, the object references in the syntax examples assume the object is being referred to via a variable called `myDocument` or `myObject` etc. In the object descriptions, the value `myDocument` is shown being assigned as a variable from the many alternative sources that you can obtain a document reference.

The DOM level specification deprecates the following properties in favor of their counterparts belonging to the `BODY` object:

- `alinkColor`
- `background`
- `bgColor`
- `fgColor`
- `linkColor`
- `vlinkColor`

The DOM level 2 specification adds the following methods:

- `importNode()`
- `createElementNS()`
- `createAttributeNS()`
- `getElementsByTagNameNS()`
- `getElementById()`

A new suite of functionality relating to the way documents are viewed is introduced at DOM level 2. This is embodied in the following classes:

- `AbstractView`
- `DocumentView`

DOM level 3 expects to add the following properties to the `document` object:

- `actualEncoding`
- `encoding`
- `standalone`
- `strictErrorChecking`
- `version`

It is also expected to add the following methods:

- `adoptNode()`
- `getElementsByAttributeValue()`

Warnings:

- There are a number of properties that are defined in both Netscape and MSIE browsers. There are also a few that each defines while the other doesn't. There is at least one (`document.title`) that is defined with different behavior in both browsers. Likewise the same is true of the support for different methods across the two browsers.
- With each release, they tend to support the extensions that the other introduced with its previous release, but they also both introduce new and incompatible extensions each time as well.

See also: `BODY` object, `Element.document`, `Form.handleEvent()`, `Frame` object, `HTML` object, JavaScript Style Sheets, `Layer.document`, `Node` object, `Node.ownerDocument`, `TextRange.queryCommandEnabled()`, `TextRange.queryCommandIndeterm()`, `TextRange.queryCommandState()`, `TextRange.queryCommandSupported()`, `TextRange.queryCommandText()`, `TextRange.queryCommandValue()`, `Window` object, `Window.document`, `Window.frames[]`, `Window.opener`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`<form_name>`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`activeElement`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
`alinkColor`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	0 +	-	Warning, Deprecated
`background`	-	-	-	-	-	0 +	-	Deprecated
`bgColor`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	0 +	-	Warning, Deprecated
`body`	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	1 +	-	ReadOnly.
`characterset`	1.5 +	-	6.0 +	-	-	-	-	-
`charset`	-	3.0 +	-	4.0 +	-	-	-	-
`cookie`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`defaultCharset`	-	3.0 +	-	4.0 +	-	-	-	Warning
`designMode`	-	5.0 +	-	5.0 +	-	-	-	Warning
`doctype`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly.
`documentElement`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	Warning, ReadOnly.
`domain`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	Warning, ReadOnly.
`expando`	-	3.0 +	-	4.0 +	-	-	-	-
`fgColor`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	0 +	-	Warning, Deprecated
`fileCreatedDate`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
`fileModifiedDate`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
`fileSize`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
`height`	1.2 +	-	4.0 +	-	-	-	-	Warning
`implementation`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	Warning, ReadOnly.

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
lastModified	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	ReadOnly.
linkColor	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	0 +	-	Warning, Deprecated
location	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning, Deprecated
parentWindow	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
protocol	-	1.0 +	-	3.02 +	-	-	-	ReadOnly.
readyState	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
referrer	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning, ReadOnly.
selection	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly.
title	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
uniqueID	-	5.0 +	-	5.0 +	-	-	-	-
URL	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly.
vlinkColor	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	0 +	-	Warning, Deprecated
width	1.2 +	-	4.0 +	-	-	-	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
attachEvent()	-	5.0 +	-	5.0 +	-	-	-	-
captureEvents()	1.2 +	-	4.0 +	-	-	-	-	Warning; Deprecated
clear()	1.0 +	1.0 +	2.0 +	3.02 +	5.0 +	-	-	Warning: Deprecated
close()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
contextual()	1.2 +	-	4.0 +	-	-	-	-	Warning, Deprecated
createAttribute()	1.5 +	-	6.0 +	-	-	1 +	-	-
createCDATASection()	1.5 +	-	6.0 +	-	-	1 +	-	-
createComment()	1.5 +	-	6.0 +	-	-	1 +	-	-
createDocumentFragment()	1.5 +	-	6.0 +	-	-	1 +	-	-
createElement()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
createEntityReference()	1.5 +	-	6.0 +	-	-	1 +	-	-
CreateProcessing Instruction()	1.5 +	-	6.0 +	-	-	1 +	-	-
createStyleSheet()	-	3.0 +	-	4.0 +	-	-	-	Warning
createTextNode()	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
detachEvent()	-	5.0 +	-	5.0 +	-	-	-	-
elementFromPoint()	-	3.0 +	-	4.0 +	-	-	-	-
execCommand()	-	3.0 +	-	4.0 +	-	-	-	Warning
getElementById()	1.5 +	5.0 +	6.0 +	5.0 +	5.0 +	1 +	-	-
getElementsByName()	1.5 +	5.0 +	6.0 +	5.0 +	5.0 +	1 +	-	Warning

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
getElementsByTagName()	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
getSelection()	1.2 +	-	4.0 +	-	-	-	-	Warning
handleEvent()	1.2 +	-	4.0 +	-	-	-	-	-
mergeAttrbiutes()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
open()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
queryCommandEnabled()	-	3.0 +	-	4.0 +	-	-	-	Warning
queryCommandIndeterm()	-	3.0 +	-	4.0 +	-	-	-	Warning
queryCommandState()	-	3.0 +	-	4.0 +	-	-	-	Warning
queryCommandSupported()	-	3.0 +	-	4.0 +	-	-	-	Warning
queryCommandText()	-	3.0 +	-	4.0 +	-	-	-	Warning
queryCommandValue()	-	3.0 +	-	4.0 +	-	-	-	Warning
recalc()	-	5.0 +	-	5.0 +	-	-	-	Warning
releaseEvents()	1.2 +	-	4.0 +	-	-	-	-	-
routeEvent()	1.2 +	-	4.0 +	-	-	-	-	-
write()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
writeln()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeCut	-	5.0 +	-	5.0 +	-	-	-	-
onBeforEditFocus	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
onBeforePaste	-	5.0 +	-	5.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onClick	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	4.0 +	Warning
onContextMenu	-	5.0 +	-	5.0 +	-	-	-	-
onCut	-	5.0 +	-	5.0 +	-	-	-	-
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDrag	-	5.0 +	-	5.0 +	-	-	-	-
onDragEnd	-	5.0 +	-	5.0 +	-	-	-	-
onDragEnter	-	5.0 +	-	5.0 +	-	-	-	-
onDragLeave	-	5.0 +	-	5.0 +	-	-	-	-
onDragOver	-	5.0 +	-	5.0 +	-	-	-	-
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onDrop	-	5.0 +	-	5.0 +	-	-	-	-
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onPaste	-	5.0 +	-	5.0 +	-	-	-	-
onPropertyChange	-	5.0 +	-	5.0 +	-	-	-	-
onReadyStateChange	-	3.0 +	-	4.0 +	-	-	-	-
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-
onStop	1.2 +	-	4.0 +	-	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Document.<form_name> (Property)

The name of a form if the document contains a `<FORM>` tag.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`Form` object	
JavaScript syntax:	-	*myDocument*`.aFormName`
HTML syntax:	`<FORM NAME="aFormName">`	
Argument list:	*aFormName*	The unique name of a form

If a `<FORM NAME="ABCD">` tag is present in the document, then there will be a property of the document object called `document.ABCD`, named after the form. This means you can access the form object directly by name.

If there are several different forms, they will each have a named property according to their names.

Warnings:

- This is not supported in the same way on MSIE.
- Be careful not to use the same name more than once. Properties must be created according to the name HTML tag attribute in the `<FORM>` tag. If there are two `<FORM NAME="ABCD">` tags, there will only be one `ABCD` property but you cannot be sure which one of the two forms will be present in it. The individual forms will still be reachable via the `forms[]` array however.

See also:	`Document` object, `Form` object, `NAME="..."`

Document.activeElement (Property)

The input element that currently has input (keyboard and mouse) focus.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`An object that can receive input focus`	
JavaScript syntax:	IE	*myDocument*`.activeElement`

This property contains a reference to the `element` object that currently has the input focus.

This may refer to an input element but if there is no active `<FORM>` in the document then the active element will be the `window` object that the document is displayed in.

See also:	`Document` object, `Frame` object, `Window` object

Property attributes:

`ReadOnly`

Document.alinkColor (Property)

The color of a link on the page while it is being activated.

Availability:	DOM level – 0 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0 Deprecated	
Property/method value type:	`Color value`	
JavaScript syntax:	-	*myDocument*`.alinkColor`
	-	*myDocument*`.alinkColor = aColorValue`
HTML syntax:	`<BODY ALINK="aColorValue">`	
Argument list:	*aColorValue*	A hex color value or color name

This value controls the text of active links in the document body. You should use the normal color values to define the required color.

This property is equivalent to the `ALINK` attribute of the `<BODY>` HTML tag. This is color that is used while the mouse is over the link and the button is held down by the user.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse. On the other hand it does work consistently on both MSIE and Netscape where style sheet controls do not.

The DOM level 1 standard deprecates the usage of this property in favor of `BODY.aLink` instead.

Warnings:

- This property can only be changed from JavaScript in the `<HEAD>` section. You cannot modify it after the `<BODY>` has commenced loading.

See also:	`BODY.aLink`, `BODY.link`, *`BODY.text`*, `BODY.vLink`, Color names, Color value, `Document` object, `Document.bgColor`, `Document.fgColor`, `Document.linkColor`, `Document.vlinkColor`

Document.all[] (Collection)

A collection object containing references to every object in the MSIE DOM.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	IE	*`myDocument`*`.all`

This is available by virtue of the document inheriting properties from an `Element` object.

Refer to the description of the `collection` object for details of how it enhances the capabilities of the basic built-in `Array` class.

Warnings:

- This function may not be available in a properly constructed level 1 DOM implementation. Therefore it may become deprecated after version 5 of MSIE.

See also:	`Collection` object, `Document` object, `Element` object, `Element.all[]`

Property attributes:

`ReadOnly`

Document.anchors[] (Collection)

An array of all the `anchor` objects in the document.

Availability:	DOM level – 1 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0
Property/method value type:	`AnchorArray object`
JavaScript syntax:	- `myDocument.anchors`
HTML syntax:	`<A>`

There is one anchor object for every `<A>` HTML tag in the page.

Prior to version 1.2 of JavaScript, this array simply contained a list of anchors but there were no anchor objects accessible until Netscape 4 implemented them. The only property that could be accessed prior to that was the length of the array.

Changing the name of an anchor in MSIE leaves the existing anchor available and adds a copy using the new name. This may be important if you are parsing the `anchors` array with an enumerator.

Refer to the description of the `AnchorArray` object for details of how it enhances the built-in `Array` class.

There is some difference between the browsers in the way they implement `anchor` and `link` objects. In MSIE the `anchor` and `link` objects are represented identically and the objects simply ignore properties they don't need. Netscape supports different objects according to whether they are links or anchors.

DOM level 1 requires that this collection contains only those objects instantiated by an `<A>` HTML tag which contain a `NAME="..."` HTML tag attribute. Any anchors that have an `ID="..."` HTML tag attribute but no `NAME="..."` attribute should not be included.

Warnings:

- Be aware that if you are not using some `NAME` or `ID` binding to the `<A>` tags, you may get unexpected results if new links are added and you are accessing elements of this array using numeric index values.
- The DOM standard mandates at level 1 that only the `A` tags that have a `NAME="..."` HTML tag attribute should be included in this collection. Anchors with an `ID="..."` HTML tag attribute but no `NAME="..."` attribute should not be included in the collection.

See also:	`Anchor` object, `Anchor.name`, `AnchorArray` object, `AnchorArray.length`, `Document` object, `Document.links[]`, `Element.all[]`, `LINK` object, `LinkArray` object

Property attributes:

`ReadOnly`

Document.applets[] (Collection)

An array containing a list of all the applets in the document.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0
Property/method value type:	`AppletArray object`
JavaScript syntax:	- `myDocument.applets`
HTML syntax:	`<APPLET>`

There is one `applet` object in this array for every `<APPLET>` tag. The applets are represented by `JavaObject` objects. These are wrappers around instances of the class `java.applet.Applet`. This is done with LiveConnect in Netscape and ActiveX in MSIE.

From Netscape 3 and MSIE version 3, your script is able to access the public methods and properties of the Java applets. Netscape supports better communication facilities between the two by virtue of its LiveConnect facility.

Every applet inherits some public properties and methods from its super-class. At least we can be certain that the applet supports the `start()` and `stop()` methods. In MSIE, the applets collection may contain references to intrinsic controls, images, embed and other non-Java objects as well. However other than those, you will need to examine the applet documentation on a case by case basis.

Refer to the description of the `AppletArray` object for details of how it enhances the built-in `Array` class.

DOM level 1 requires that this collection includes objects that are instantiated by `<OBJECT>` HTML tags. It also notes that `<APPLET>` HTML tags should be deprecated.

Warnings:

- Be aware that if you are not using some `NAME` or `ID` binding to the `<APPLET>` tags, you may get unexpected results if new applets are added and you are accessing elements of this array using numeric index values.
- In MSIE version 4, this returns a `Collection` object instead of an `AppletArray` object. MSIE 5 is more consistent with Netscape .

See also:	`Applet` object, `AppletArray` object, `Document` object, `Document.embeds[]`, `JavaObject` object, LiveConnect, `OBJECT` object

Property attributes:

`ReadOnly`

Document.attachEvent() (Method)

A means of attaching events to windows and documents.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myDocument*`.attachEvent(`*anEventName*`,` *anEventHandler*`)`
Argument list:	*anEventHandler*	A reference to an event handler function
	anEventName	The name of an event to be handled

This is part of the behavior handling in MSIE which involves the use of style sheets and `.htc` files. It is a way of binding a function to an event so that when the event fires, the function is called. It can be applied in a more general way than just with behaviors.

See also:	`<STYLE>`, `Document.detachEvent()`, `Window.attachEvent()`, `Window.detachEvent()`

Document.background (Property)

DOM originally intended this to be the URL of a background image for the current document.

Availability:	DOM level – 0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myDocument*`.background`

The background image for the document which is defined in the `<BODY>` tag is actually stored in the `BODY` object in MSIE. You cannot access the background image directly in Netscape because the `BODY` object is not available.

DOM level 1 deprecates the use of this property in favor of the `BODY.background` property. Accessing this property in MSIE and Netscape returns the undefined value.

See also:	BODY.background

Document.bgColor (Property)

The background color of the document.

Availability:	DOM level – 0 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0 Deprecated
Property/method value type:	`color value`
JavaScript syntax:	- *myDocument*.bgColor - *myDocument*.bgColor = *aColorValue*
HTML syntax:	`<BODY BGCOLOR="aColorValue">`
Argument list:	*aColorValue* A hex color value or color name

This corresponds to the `BGCOLOR="..."` HTML tag attribute on the `<BODY>` tag.

You can modify this value at any time, the results of which will be to change the background color of the page.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse. On the other hand it does work consistently on both MSIE and Netscape where style sheet controls do not.

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image, in case the image takes a long time to load or the browser is unable to display a background image.

Warnings:

- On the Unix platform, the Netscape browser versions 2 and 3 exhibit a bug when changing the background color. If you attempt to do this, the page content disappears until something causes the window to be redrawn. If you are using X-Windows, you may be able to alleviate this by changing the damage control policy; however, you would likely need to change it to a slower performing damage control model and that change would apply to all applications in the session. Alternatively, detect the platform and avoid the background color change.

See also:	`BODY` object, `BODY.aLink`, `BODY.bgColor`, `BODY.link`, `BODY.text`, `BODY.vLink`, Color names, Color value, `Document` object, `Document.alinkColor`, `Document.fgColor`, `Document.linkColor`, `Document.vlinkColor`, `style.backgroundColor`

Document.body (Property)

The contents of the <BODY> tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`BODY` object	
JavaScript syntax:	-	`myDocument.body`

In MSIE, this property is a reference to a `BODY` object. Early versions of Netscape are not DOM compliant and do not support this property since it has no `BODY` object implemented to refer to. This is corrected in Netscape 6.0 which implements full DOM level 1 compliance.

The DOM level 1 specification states that this property should yield an `HTMLElement` object. A `BODY` object is derived from an HTMLElement so that general rule is satisfied.

If the document is a frameset then it will return a `FRAMESET` object.

See also:	`BODY` object, `Document` object, `Window.pageXOffset`, `Window.pageYOffset`

Property attributes:

`ReadOnly`

Document.captureEvents() (Function)

Part of the Netscape 4 event propagation complex.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myDocument`*`.captureEvents(`*`anEventMask`*`)`
Argument list:	*`anEventMask`*	A mask constructed with the manifest event constants

This is part of the event management suite which allow events to be routed to handlers other than just the one that defaults to being associated with an event.

The events to be captured are signified by setting bits in a mask.

This method allows you to specify what events are to be routed to the receiving `Document` object.

The events are specified by using the bitwise `OR` operator to combine the required event mask constants into a mask that defines the events you want to capture. Refer to the Event Type Constants topic for a list of the event mask values.

A limitation of this technique is that ultimately, only 32 different kinds of events can be combined in this way and this may limit the number of events the browser can support. Since this is only supported by Netscape, the functionality is likely to be deprecated when the standards bodies agree on a standard way of handling events. Then we simply need to wait for the browser manufacturers to support the standardized behavior.

In the meantime, we shall have to implement scripts using this capability if we need to build complex event handling systems. A different script will be required for MSIE.

You may be able to factor your event handler so that you only have to make platform specific event dispatchers and can call common handling routines that can be shared between MSIE and Netscape.

The example copes with cross browser execution in an interesting way (note; this doesn't work for Opera or N 6).

Warnings:

- Since a bit mask is being used, this must be an int32 value. This suggests that there can only be 32 different `Event` types supported by this event propagation model.
- This capability is deprecated and is not supported in Netscape 6.0 anymore. It never was supported by MSIE which implements a completely different event model. As it turns out the DOM level 2 event model converges on the MSIE technique.

Example code:

```
// A portable keyboard eventhandler
// Provided by Jon Stephens
function handleKeypress(event)
{
   var key;
   if(document.layers)
   {
      key = event.which;
   }
   if(document.all)
   {
      event = window.event;
      key = event.keyCode;
   }
   alert("Key: " + String.fromCharCode(key) + "\nCharacter code: " + key + ".");
}

if(document.layers)
{
   document.captureEvents(Event.KEYPRESS);
}

document.onkeypress = handleKeypress;
```

See also:	`captureEvents()`, `Document` object, `Document.releaseEvents()`, `Element.onevent`, Event management, Event propagation, Event type constants, `Frame` object, `Layer.captureEvents()`, `Layer.releaseEvents()`, `onMouseMove`, `Window` object, `Window.captureEvents()`, `Window.releaseEvents()`

Document.characterset (Property)

A Netscape 6.0 equivalent of the `Document.charset` property.

Availability:	JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myDocument*`.characterset`

Refer to:

`Document.charset`

Document.charset (Property)

The character set currently being used.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.charset`

This would contain the character set being used by the document. For example the value `"iso-8859-1"` is likely to be returned but the local variant of the browser and OS may affect the value you get.

This property might contain a value such as:

```
csISO5427Cyrillic
```

Details of other aliases can be located at the IANA registry.

See also:	`Document` object, `Document.defaultCharset`

Web-references:

ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

Document.classes[] (Collection)

Part of the JSS style control model supported only by Netscape .

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated
Property/method value type:	`JSSClasses` object

This returns a collection of JSS classes. These are associative arrays that contain other associative arrays. They don't inherit properties and methods from the `Array` object class and are particularly difficult to operate on.

If you are working with JSS driven style sheets, for a start your project is not portable across browser. It will only work on Netscape. Secondly, you are using deprecated functionality. This will lead to your project code breaking at some time in the future when JSS support is no longer available.

Warnings:

- This functionality is removed from Netscape 6.0.

See also:	`Document` object, Document.contextual(), `Document.ids[]`, `Document.tags[]`,`JSSClasses` object

Property attributes:

ReadOnly.

Document.clear() (Method)

A deprecated method that clears the document.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 5.0 Deprecated	
JavaScript syntax:	-	`myDocument.clear()`

This is a method that was implemented early in the life of the Netscape browser. It is only provided for backwards compatibility with older projects and should not be used in any new work you do.

Warnings:

- This method is deprecated as of JavaScript version 1.2 from which point you simply need to use `document.open()` to create a fresh (and empty) document.

See also:	`Document` object

Document.close() (Method)

Close a document body after you have finished writing to it.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
JavaScript syntax:	-	*myDocument*.close()

Calling the `document.close()` method will cause the output buffer to be flushed, stop the rotating loading icon and force the display of any HTML you have written. In some browsers this may happen automatically when the script exits and in others you will need to do it explicitly. It is good coding style to close a stream when you have finished writing to it. This is analogous to the `fflush()` call that C programmers make to force the buffered output to be sent through an I/O stream.

If you close a document stream and then proceed to write to it again, the document will be implicitly cleared and reopened. That is as if you had performed a `document.clear()` and then a `document.open()`. Any content that you had just written and then terminated with the `document.close()` will be discarded and a new document body started.

The DOM level 1 specification suggests that this method may be deprecated in the future.

See also:	`Document` object, `Document.open()`, `Document.write()`, `Document.writeln()`

Document.contextual() (Method)

Returns a style object that represents the contextual style for the receiver.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated	
Property/method value type:	`JSSTag` object	
JavaScript syntax:	N	*myDocument*.contextual(*aStyle, ...*)
Argument list:	*aStyle*	One or more style objects to clone

This method is used to instantiate a new `style` object for when tags occur within the tags or classes of other types. For example, you can build a contextual `style` object that combines two tag types and specifies that when, and only when, they are used in that context, a certain style appearance is defined.

This fragment of JavaScript will yield a `style` object that is relevant to the contents of an `<H1>` tag:

```
myStyle1 = document.tags.H1;
```

This fragment will yield a `style` object relevant to text inside a `<B>` tag (bold text):

```
myStyle2 = document.tags.B;
```

Now we can combine the two to yield a `style` object that applies only to content between the `<B>` tags when they are inside `<H1>` tags:

```
myStyle3 = document.contextual(myStyle1, myStyle2);
```

Finally, we can set some attribute of that style:

```
myStyle3.color = "Blue";
```

So, any text inside `<B>` tags inside `<H1>` tags will be blue.

This needs to be executed in a `<SCRIPT>` block in the `<HEAD>` portion of the document and cannot be executed after the body has commenced loading.

The result of this method call is a `style` object that operates within a particular context.

Warnings:

- Because this is part of the JSS support, its use is deprecated and likely to become redundant now that Netscape is orienting its style sheet capabilities towards the open standards from W3C.

See also:	`Document` object, `Document.classes[]`, `Document.tags[]`, `JSSTag` object

Document.cookie (Property)

Access to a cookie for the current document.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myDocument*.cookie

A cookie is a small fragment of textual data that is associated with the current page. It is not modelled particularly well for access by JavaScript and you will need to use some scripting to disassemble and reassemble the name-value pairs that are concatenated together to make the cookie.

The `cookie` property for a document returns a string containing ALL the cookies that apply to the document. You will need to split them into individual cookies by separating them at semi-colon boundaries. From that you will for each cookie obtain a `name=value` construct that you can further dismantle and process.

Note that you will not get any of the special attributes of the cookie since they are write-only. The only thing you can get back is its `value` property. This means that although the `cookie` property is available for read and write access, it is unlike all other properties it is not symmetrical. You don't get back out what you put in.

Warnings:

- MSIE version 3 will only return cookie data for documents that were requested using the `http:` protocol.

Example code:

```
// Define a cookie for the current document
document.cookie = "cookiename=value";
// Define a cookie from a variable using URL escape()
document.cookie = "cookiename=" + escape(myValue);
// Setting an expiry date on the cookie
myCookieValue   = "cookiename=value";
myCookieExpires = "expires="+myDate.toGMTString();
myCookie = myCookieValue + "; " + myCookieExpires;
```

See also:	Cookie, `Document` object

Document.createAttribute() (Method)

Creates an `Attribute` object that can then set on an Element with the `setAttributeNode()` method.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`Attribute` object	
JavaScript syntax:	N	*myDocument*`.createAttribute(`*aName*`)`
Argument list:	*aName*	The name of the attribute to create

This is a new feature introduced with the DOM level 1 standard and currently available only in fully DOM compliant browsers.

The example function shows how a new `attribute` object can be created and set on an element. Because this is standardized at the DOM document level and is not dependent on HTML, it could work in non-web browser implementations.

This is where browser implementations of DOM functionality tend to blur the facts slightly. DOM describes a generic `Document` and then sub-classes that to describe an `HTMLDocument`. Browsers merge the two areas of functionality and simply call it a `Document`.

Example code:

```
// A code fragment that creates an attribute and sets it on
// an element object that is passed in:
function attachMyAttrib(anElement, aName, aValue)
{
var myNewAttr = createAttribute(aName);
var myOldAttr = anElement.setAttributeNode(myAttr);
}
```

See also:	`Attr` object, `Attribute` object

Document.createCDATASection() (Method)

Creates a new `CDATASection` object whose value is the string passed as an argument.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0
Property/method value type:	`CDATASection` object
JavaScript syntax:	N *myDocument*`.createCDATASection(someData)`
Argument list:	*someData* The data content for the new object

See also:	`CDATASection` object

Document.createComment() (Method)

Creates a new comment `node` object containing the data passed in the string argument.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0
Property/method value type:	`COMMENT` object
JavaScript syntax:	N *myDocument*`.createComment(`*someData*`)`
Argument list:	*someData* The content of the comment block

See also:	`COMMENT` object

Document.createDocumentFragment() (Method)

Creates a new and empty document fragment.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`DocumentFragment` object	
JavaScript syntax:	N	*myDocument*`.createDocumentFragment(`*someData*`)`
Argument list:	*someData*	The content of the document fragment

See also:	`DocumentFragment` object

Document.createElement() (Method)

A method to create a new element within a document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Element` object	
JavaScript syntax:	-	*myDocument*`.createElement(`*aTagName*`)`
Argument list:	*aTagName*	An HTML tag name

The result of calling this method is a new `Element` object as if it had been freshly instantiated by an HTML tag.

This is a means of creating new objects which you can then assign to various properties. It is effectively a constructor which is driven by the HTML tag names.

Internally, the MSIE browser maintains objects that are associated with HTML tags as members of a class named after the tag that created them. It's as if there were a class that corresponded to each HTML tag. This means that a call like `createElement()` can use them as if they were constructor objects.

The internal mechanisms in Netscape 6.0 follow the DOM level 1 specification more closely and use the correct class names as defined in the standards. MSIE may support these in the future and you should be careful when writing any script that needs to be aware of the object class names it is operating on.

See also:	`Document` object, `Element` object

Document.createEntityReference() (Method)

A new `EntityReference` object is created. It may acquire the same child list as the entity it refers to.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`EntityReference` object	
JavaScript syntax:	N	*myDocument*`.createEntityReference(`*aName*`)`
Argument list:	*aName*	The name of the entity reference to be created
See also:	*EntityReference* object	

Document.createProcessingInstruction() (Method)

A new processing instruction node is created. Its name and content are specified by the arguments.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`ProcessingInstruction` object	
JavaScript syntax:	N	*myDocument*`.createProcessingInstruction (`*aTarget*`,`*someData*`)`
Argument list:	*aTarget*	The target for the instruction
	someData	The data for the instruction
See also:	`ProcessingInstruction` object	

Document.createStyleSheet() (Method)

A style sheet factory method.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`styleSheet` object	
JavaScript syntax:	IE	*myDocument*`.createStyleSheet()`
	IE	*myDocument*`.createStyleSheet(`*aURL*`)`
	IE	*myDocument*`.createStyleSheet(`*aURL*`, `*anIndex*`)`
Argument list:	*aURL*	The URL to load the style sheet from
	anIndex	A location within the stylesheet list to insert this stylesheet

This is a means of adding a style sheet to a document from the JavaScript context.

When you create the style sheet, you have the choice of simply creating an empty `styleSheet` object or calling one in from a URL document source location.

You can also specify where in the hierarchy of currently loaded style sheets this new one should be placed by specifying its index location within the styleSheets collection.

If you can locate this `styleSheet` object once it has been installed, you can carry out further modifications on it by means of the `addRule()` method. This may be somewhat problematic on some browser versions and platforms since you don't get an object reference handle back from this method. You might be able to code around that, however, since you can specify a target index position in the `styleSheets` collection. That may mean you can get a reference to it indirectly once it has been added.

Warnings:

- Although this method is supposed to return a `styleSheet` object, on the Macintosh platform it returns a `null` value in some versions of the MSIE browser.

See also:	`Document` object, `StyleSheet` object, `StyleSheet.addRule()`

Document.createTextNode() (Method)

A means of constructing a new `textNode` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`textNode` object	
JavaScript syntax:	-	*myDocument*`.createTextNode(`*someData*`)`
Argument list:	*someData*	The textual content of the node

You may need to create additional text nodes to be placed between other objects you create when modifying a document object model. Text nodes are placed interstitially between object nodes within a document hierarchy.

See also:	Hierarchy of objects, `textNode` object

Document.defaultCharset (Property)

The default character set of the document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.defaultCharset`

This would contain the character set being used by the document when it was first opened. That may have changed but this value should always be the same as it was at the start. For example the value `"iso-8859-1"` is likely to be returned but the local variant of the browser and OS may affect the value you get.

Warnings:

- Netscape does not support this.

See also:	`Document` object, `Document.charset`

Document.designMode (Property)

Part of a page authoring control system built into the MSIE browser.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myDocument*`.designMode`

This is a means of editing objects manually within the browser. While the browser is in this state, editable elements acquire a UI for modification by means of the `enter` key. When the `designMode` property is set to `"on"` you cannot execute scripts.

The following values can be assigned to this property:

- `On`
- `Off`
- `Inherited`

Warnings:

- Whether this works may depend on platform and browser versions. It does not appear to be functional on MSIE for Macintosh despite the Microsoft documentation saying that it should work.

Document.detachEvent() (Method)

A means of detaching events from windows and documents that were previously attached with the `attachEvent()` method.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myDocument*.detachEvent(anEventName)
Argument list:	*anEventName*	The name of an event to be handled

This is part of the behavior handling in MSIE which involves the use of style sheets and `.htc` files.

See also:	<STYLE>, Document.attachEvent(), Window.attachEvent(), Window.detachEvent()

Document.doctype (Property)

The current document type of the document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	Doctype object	
JavaScript syntax:	-	*myDocument*.doctype
HTML syntax:	<!DOCTYPE aDocumentDescription>	
Argument list:	*aDocumentDescription*	A reference to a DTD for this document

The <!DOCTYPE> tag at the top of the document is instantiated into a `Doctype` object and a reference to it is stored here. This tag describes a DTD statement that identifies the type of document.

Here is a properly formed DTD statement:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 FINAL//EN">
```

This is represented by an object whose type is a single exclamation mark. As this tag should be on the first line of a document, the object will be at index 0 in the `document.all[]` array.

Support for this is still somewhat patchy as of Netscape 6.0 and MSIE 5.5. Netscape 6.0 returns a `DocType` object (note the class name capitalization) but its string value is not defined. MSIE returns an undefined value.

Warnings:

- There is apparently no access to this value in Netscape and MSIE only provides limited access.

See also:	`!` object, `!.tabIndex`, `Doctype` object, `Document` object, `Element` object

Property attributes:

`ReadOnly`

Document.documentElement (Property)

An HTML element that represents the document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`Element` object
JavaScript syntax:	- *myDocument*`.documentElement`
HTML syntax:	`<HTML>`

This property contains a reference to an `Element` object that corresponds to the `<HTML>` tag at the top of the document.

Warnings:

- Traversing this object with a `for( ... in ... )` loop in MSIE 5 may lead to a browser crash. This is unfortunately still true for version 5.5 of MSIE.

See also:	`Document` object, `Element` object, `HTML` object

Property attributes:

`ReadOnly`

Document.domain (Property)

A means of allowing web servers that trust one another to allow normally insecure access from one another's documents.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myDocument`*`.domain`

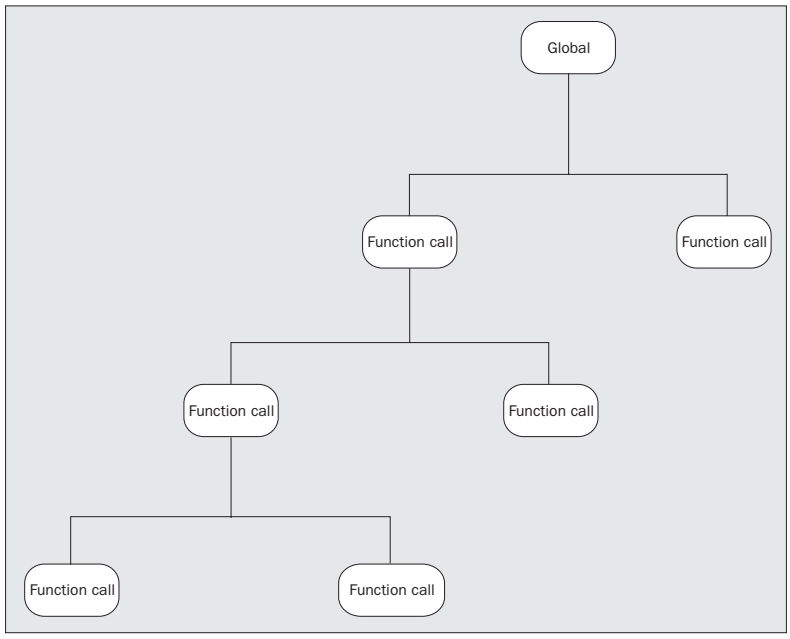
This is part of the Netscape security model that allows unsigned scripts to communicate with one another as long as they belong to the same domain.

Normally, documents may not always be able to communicate with one another because they come from different servers. This can be alleviated by specifying a domain that is shared by several servers.

The initial value of this property is the complete host name of the server that provided the document when it was loaded.

A script is able to set a domain value that is a suffix of the existing domain. So, `aaa.bbb.ccc.com` can become `bbb.ccc.com` but cannot become `another.domain.com` which allows you to serve documents from several hosts in the same domain but not access content served by other domains.

Once any two documents have the same domain setting, they can exchange property values with one another even though they may have originated on different hosts.



Warnings:

- This is not supported on the WebTV platform.

See also:	`Document` object, JellyScript, Security policy

Property attributes:

`ReadOnly`

Document.elementFromPoint() (Method)

Determines which element is under a particular `x, y` location.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Object` object	
JavaScript syntax:	IE	*myDocument*`.elementFromPoint(`*xCoordinate*`,` *yCoordinate*`)`
Argument list:	*xCoordinate*	The X-coordinate value
	yCoordinate	The Y-coordinate value

The result of this method will be a reference to the `Element` object under the `x,y` point.

The document is inspected and the browser works out the topmost `Element` object at the indicated `x,y` location. That `Element` object is then returned as the result. This is very much like executing a mouse click at a location in the document window, and then extracting the target `Element` object from the `event` object. However this is far simpler.

See also:	`Document` object

Document.embeds[] (Collection)

An array of all the `<EMBED>` tag objects within the document.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`EmbedArray` object	
JavaScript syntax:	-	*myDocument*`.embeds`
HTML syntax:	`<EMBED>`	

Each <EMBED> tag is represented here by an object. The embeds are encapsulated in the same JavaObject objects as are used for the Java applets.

Every embedded plugin will respond to different suites of property and method messages although there may be some similarities between some plugins that serve the same purpose.

Warnings:

- Be aware that if you are not using some NAME or ID binding to the <EMBED> tags, you may get unexpected results if new embeds are added and you are accessing elements of this array using numeric index values.
- In Netscape, your JavaScript code interacts with them courtesy of LiveConnect in a very similar way. MSIE interacts with the embeds by means of ActiveX. This can lead to some differences in the way the plugins are supported and the things they can do in each browser environment.
- In MSIE on the Windows platform, plugins are often recommended for use with the <OBJECT> tag rather than the <EMBED> tag. This can lead to portability issues in your scripts and web pages.
- In MSIE version 4, this returns a Collection object instead of an EmbedArray object. MSIE 5 is more consistent with Netscape .
- Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the EMBED class. In Netscape they are objects commonly referred to as belonging to the Plugin class although they are really implemented as JavaObject objects. In MSIE, this is an ActiveX object.
- There is additional confusion in that there is a plugins[] array that belongs to the document and another that belongs to the navigator object. They both contain collections of objects but of different types. This is further confused by the fact that the document.plugins[] array is another name for the document.embeds[] array.
- Because of this confusing situation, the best recommendation is that we refer to document.embeds[] and navigator.plugins[] and quietly ignore the document.plugins[] array. Furthermore we shall refer to Plugin objects as being something the browser can use to play embedded content and Embed objects will be an instance of a plugin that is alive and running in a document.

See also:	<EMBED>, Document object, Document.applets[], Document.plugins[], Embed object, EmbedArray object, JavaObject object, LiveConnect

Property attributes:

ReadOnly

Document.execCommand() (Method)

Part of an MSIE special document command handling mechanism. A method for executing commands.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	Boolean primitive

JavaScript syntax:	IE	*myDocument*.execCommand(*aCommand*)
	IE	*myDocument*.execCommand(*aCommand*, *aUIFlag*)
	IE	*myDocument*.execCommand(*aCommand*, *aUIFlag*, *aParameter*)

Argument list:	*aCommand*	An MSIE command to execute
	aParameter	Parameter value to the command
	aUIFlag	Display or inhibit UI appearance

The MSIE browser supports a special command handling interface that hooks through the browser's user interface. It allows you to automate user actions in a way that other browsers and non-Windows platform users cannot take advantage of.

Although this is a method that applies to a `document` object, many of the commands that are executed through this mechanism will require that a `TextRange` object is created and available first.

The result returned by this method, is a Boolean `true` if the action succeeded, and a Boolean `false` if it failed in someway.

The flag parameter provides a way to suppress any user interface changes that may appear as a result of executing the command.

Name	Description
`2D-Position`	Absolutely positioned elements can be moved by dragging.
`AbsolutePosition`	Sets an element's position property to "`absolute`"
`BackColor`	The background color for the current selection is set to the color value passed in the parameter argument.
`Bold`	The selected text has `<B>` and `</B>` tags placed at either end.
`Copy`	The `TextRange` is copied to the clipboard.
`CreateBookmark`	Carries out modifications to an existing `<A>` tag or creates one, then adds the item to the bookmarks list. The parameter provides the `NAME` value. The `<A>` tag is removed if there is no parameter.
`CreateLink`	Wraps an `<A HREF="...">` tag around the selected text. The parameter contains the URL value for the HREF.
`Cut`	Performs a cut to clipboard.
`Delete`	The text range is deleted. This is not the same as a `Cut` command.
`FontName`	Wraps `<FONT>` tags round the selection. The required font face is passed in the parameter.
`FontSize`	Wraps `<FONT>` tags round the selection and defines the fonts size from the parameter value.
`ForeColor`	Redefines the foreground `btext` color for the selection taking the color value from the parameter.
`FormatBlock`	Wraps a `<BLOCK>` tag round the `TextRange`.
`Indent`	The `TextRange` is indented
`InsertButton`	A `<BUTTON>` tag is placed at the current insertion point in the document. Its `ID` value is defined by the parameter.

Table continued on following page

Name	Description
InsertFieldset	A <FIELDSET> tag is inserted with the ID value being taken from the parameter
InsertHorizontalRule	An <HR> tag is added at the current insertion point.
InsertIFrame	A new <IFRAME> is inserted with the content URL being provided in the parameter.
InsertImage	Overwrites an image on the current selection.
InsertInputButton	An <INPUT TYPE="Button"> is added with its ID value coming from the parameter.
InsertInputCheckbox	An <INPUT TYPE="Checkbox"> is added with its ID value coming from the parameter.
InsertInputFileUpload	An <INPUT TYPE="FileUpload"> is added with its ID value coming from the parameter.
InsertInputHidden	An <INPUT TYPE="Hidden"> is added with its ID value coming from the parameter.
InsertInputImage	An <INPUT TYPE="Image"> is added with its ID value coming from the parameter.
InsertInputPassword	An <INPUT TYPE="Password"> is added with its ID value coming from the parameter.
InsertInputRadio	An <INPUT TYPE="Radio"> is added with its ID value coming from the parameter.
InsertInputReset	An <INPUT TYPE="Reset"> is added with its ID value coming from the parameter.
InsertInputSubmit	An <INPUT TYPE="Submit"> is added with its ID value coming from the parameter.
InsertInputText	An <INPUT TYPE="Text"> is added with its ID value coming from the parameter.
InsertMarquee	A new <MARQUEE> is added with the ID being taken from the parameter.
InsertOrderedList	A new <OL> is added with the ID being taken from the parameter.
InsertParagraph	A new <P> is added with the ID being taken from the parameter.
InsertSelectDropdown	A new <SELECT TYPE="select-one"> is added with the ID being taken from the parameter.
InsertSelectListbox	A new <SELECT TYPE="select-multiple"> is added with the ID being taken from the parameter.
InsertTextArea	A new <TEXTAREA> is added with the ID being taken from the parameter.
InsertUnorderedList	A new <UL> is added with the ID being taken from the parameter.
Italic	The TextRange is enclosed with <I> tags.
JustifyCenter	The TextRange is centered within its parent object.
JustifyFull	The TextRange is fully justified.
JustifyLeft	The TextRange is left justified.
JustifyRight	The TextRange is right justified.

Table continued on following page

Name	Description
LiveResize	Causes the MSHTML Editor to update an element's appearance continuously during a resizing or moving operation, rather than updating only at the completion of the move or resize.
MultipleSelection	Allows for the selection of more than one site selectable element at a time when the user holds down the SHIFT or CTRL keys.
Outdent	The complement of the Indent command.
OverWrite	The input-typing mode is set to overwrite if the parameter value is true and insert if it is false.
Paste	The contents of the clipboard are pasted into the TextRange.
PlayImage	If an image represents a video clip, then it starts playing.
Refresh	The document is reloaded.
RemoveFormat	The complement of the FormatBlock command.
SaveAs	Saves the current Web page to a file.
SelectAll	The entire document text is selected.
StopImage	The complement of the PlayImage command.
UnBookmark	The complement of the CreateBookmark command.
Underline	Places <U> tags around the TextRange.
Unlink	The complement of the CreateLink command.
Unselect	Unselects whatever was selected to create the TextRange. Many commands are now inappropriate until a new TextRange has been created.

None of these commands provide any greatly significant functionality as far as dynamic HTML is concerned. A few of them allow you to manage the clipboard and bookmark lists. It is probably best to avoid using these commands and use the more usual ways of accessing the document internals.

Warnings:

- This is only supported by the 32 bit Windows version of MSIE.

See also:	Document object, Document.queryCommandEnabled(), Document.queryCommandIndeterm(), Document.queryCommandState(), Document.queryCommandSupported(), Document.queryCommandText(), Document.queryCommandValue(), FileUpload.select(), TextRange.execCommand()

Document.expando (Property)

A means of locking objects to prevent new properties being added.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Property/method value type:	Boolean primitive	
JavaScript syntax:	IE	*myDocument*.expando
	IE	*myDocument*.expando = aSwitch
Argument list:	*aSwitch*	A boolean value to turn this feature on or off

When the `document.expando` property is set to `false`, it inhibits the creation of new properties if they do not already exist. This can sometimes help find bugs in your scripts. Imagine that you may have misspelled a property name. Setting this property so that it generates a runtime error when a new property is created may help you to isolate the fault.

See also:	`Document` object

Document.fgColor (Property)

The foreground color for text in the current document.

Availability:	DOM level – 0 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0 Deprecated	
Property/method value type:	Color value	
JavaScript syntax:	-	*myDocument*.fgColor
	-	*myDocument*.fgColor = aColorValue
HTML syntax:	<BODY TEXT="aColorValue">	
Argument list:	*aColorValue*	A hex string or color name

This value controls the foreground text in the document body. You should use the normal color values to define the required color.

This is the default text color for the document. It corresponds to the `TEXT` attribute in the `<BODY>` tag.

Default foreground text is colored according to this setting unless it is in an `<A>` tag when the `alinkColor`, `linkColor` and `vlinkColor` values override it. The foreground text color can be changed inline with the `<FONT COLOR="...">` HTML tag attribute.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse. On the other hand it does work consistently on both MSIE and Netscape where style sheet controls do not.

DOM level 1 deprecates the use of this property in favor of the `BODY.text` property.

Warnings:

- This property can only be changed in the `<HEAD>` section. You cannot modify it after the `<BODY>` has commenced loading.

See also:	BODY.aLink, BODY.link, BODY.text, BODY.vLink, Color names, Color value, Document object, Document.alinkColor, Document.bgColor, Document.linkColor, Document.vlinkColor

Document.fileCreatedDate (Property)

The date that the document file was created.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myDocument*.fileCreatedDate

Where it is possible to distinguish properties of document source files, this can tell you about the history of the document.

This read-only property describes the date that a file was created. With this you can calculate the age of the file by subtracting that date from the date and time now.

See also:	IMG.fileCreatedDate

Property attributes:

ReadOnly

Document.fileModifiedDate (Property)

The date that the document file was last modified.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myDocument*.fileModifiedDate

Where it is possible to distinguish properties of document source files, this can tell you about the history of the document.

This read-only property describes the date that a file was last modified. With this you can calculate the age of the file content by subtracting that date from the date and time now.

Because this is MSIE specific functionality, you should use the Document.lastModified property for portable script code.

See also:	Document.lastModified, IMG.fileModifiedDate

Property attributes:

`ReadOnly`

Document.fileSize (Property)

The size in bytes of the file that was received by the browser.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myDocument*`.fileSize`

This is the exact length of the received HTTP body portion of the document. This does not count any HTTP headers that the web server may have sent prior to the HTTP body. If you edit a document in a text file and then load it into the web browser, this is the exact length of that text file in characters.

On the Macintosh operating system, some text editors hide additional resource data in the file. This is not included and on that platform, the `fileSize` property is a measurement of the data fork of the file and does not include the resource fork.

See also:	`Document` object, `IMG.fileSize`

Property attributes:

ReadOnly

Document.forms[] (Collection)

An array containing a list of all the forms in the document.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 3.0 Internet Explorer – 4.0 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`FormArray` object	
JavaScript syntax:	-	`myDocument.forms`
HTML syntax:	`<FORM>`	

Every `<FORM>` element in the document corresponds to an object in the `forms[]` array.

Warnings:

- Be aware that if you are not using some `NAME` or `ID` binding to the `<FORM>` tags, you may get unexpected results if new forms are added and you are accessing elements of this array using numeric index values.
- In MSIE version 4, this returns a generic untyped object instead of an `AppletArray` object. MSIE 5 is more consistent with Netscape .
- Following some experiments, it seems that in Netscape (version 4.75) the length of the `forms[]` array indicates only the number of numerically indexed items in the array. There are additional items added to the array to refer to the same `Form` objects by name.

Example code:

```
// Referring to the first form of a document
myFirstForm = document.forms[0];
// Referring to the last form of a document
myLastForm = document.forms[document.forms.length-1];
```

See also:	`Document` object, `Element.all[]`, `Form` object, `FormArray` object

Property attributes:

`ReadOnly`

Document.frames[] (Collection)

An array containing references to all the frame objects within a document.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Frames` object	
JavaScript syntax:	IE	*myDocument*`.frames`
HTML syntax:	`<FRAME>`	

In the MSIE browser, you can define in-line frames with the `<IFRAME>` tag. The `frame` property of a document contains a reference to a `Frames` object which is a collection of `Frame` or `Window` objects that enumerate the child frames within the current document.

Aside from the fact that the frame inlined into the HTML and can appear like an image embedded in the text, in all other respects it is like `<FRAME>` in a `<FRAMESET>` and is instantiated as a `Window` object.

To locate the objects representing these in-line frame, use the normal DOM navigation techniques based on object `ID` values and collections belonging to the document.

Warnings:

- Be careful not to confuse this property with the `Window.frames` property.

See also:	`Document` object, `Frame` object, `Frames` object, `Window` object, `Window.frames[]`

Property attributes:

`ReadOnly`

Document.getElementById() (Method)

An accessor method for retrieving objects from within the DOM hierarchy specifically according to their `ID` value.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`Element` object	
JavaScript syntax:	-	*myDocument*`.getElementById(anID)`
Argument list:	*anID*	The ID value of an object to locate in a string

Sometimes, it can be inconvenient to walk the document hierarchy to locate an object by its `ID`. One alternative is to search the `all[]` collection on the MSIE browser but this is a linear search that takes place serially through the document object collection from beginning to end. It can take quite a while to locate the object.

There may also be some conflict in locating objects. The `ID` and `NAME` attributes are different and yet often the two namespaces are combined and searched together.

This method is a faster way of locating object specifically by its `ID` value and ignoring its NAME value. It searches the only unmerged namespace for the `ID` value.

The DOM level 1 specification notes that the behavior is undefined if more than one item shares the same `ID`. Running a short test on MSIE on the Macintosh platform yields the first occurring element with a matching `ID` value. It is likely that this is the generic behavior for versions of MSIE across all other platforms too. For now at least Netscape 6.0 seems to do the same thing.

As there is undefined behavior in the specification, you may find that some browsers return a different object. Indeed, because this is undefined, the behavior may change from one browser version to the next.

Document.getElementsByName() (Method)

An accessor method for retrieving objects from within the DOM hierarchy specifically according to their `NAME` value.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`NodeList` object	
JavaScript syntax:	-	*myDocument*`.getElementsByName(`*aName*`)`
Argument list:	*aName*	The name of the element to be retrieved

Sometimes, it can be inconvenient to walk the document hierarchy to locate an object by its `NAME`. One alternative is to search the `all[]` collection but this is a linear search that takes place serially through the document object collection from beginning to end. It can take quite a while to locate the object.

There may also be some conflict in locating objects. The `ID` and `NAME` attributes are different and yet often the two namespaces are combined and searched together.

This method is a faster way of locating an object specifically by its `NAME` value and ignoring its `ID` value. It searches the only unmerged namespace for the `NAME` value.

Warnings:

- This always yields an empty collection on MSIE 5 for Macintosh.

Document.getElementsByTagName() (Method)

A node list is returned that contains references to all the child elements having the specified tag name.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`NodeList` object	
JavaScript syntax:	-	*myDocument*`.getElementsByTagName(`*aTagName*`)`
Argument list:	*aTagName*	The name of an HTML tag

See also:	`Element.getElementsByTagName(),NodeList` object

Document.getSelection() (Method)

Return the currently selected text string.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myDocument*`.getSelection()`

If you have selected a piece to text, in Netscape this method will return the raw unformatted text within the selection. If there are any HTML tags in the selection, they will be stripped out.

The MSIE browser employs a completely different technique that involves `Selection` objects and `TextRange` objects as a means of access to the selected text.

As it is easy to deselect the highlighted text by clicking on some other active object in the page, you will need to access the selection inside an event handler that is triggered by the selection action itself. This might be done quite effectively in an `onSelectStart` handler.

Warnings:

- On MSIE, use the selection property of the document object.

See also:	`Document` object, `Document.selection`, `Selection` object

Document.handleEvent() (Function)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myDocument*`.handleEvent(`*anEvent*`)`
Argument list:	*anEvent*	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`Document` object, `handleEvent()`

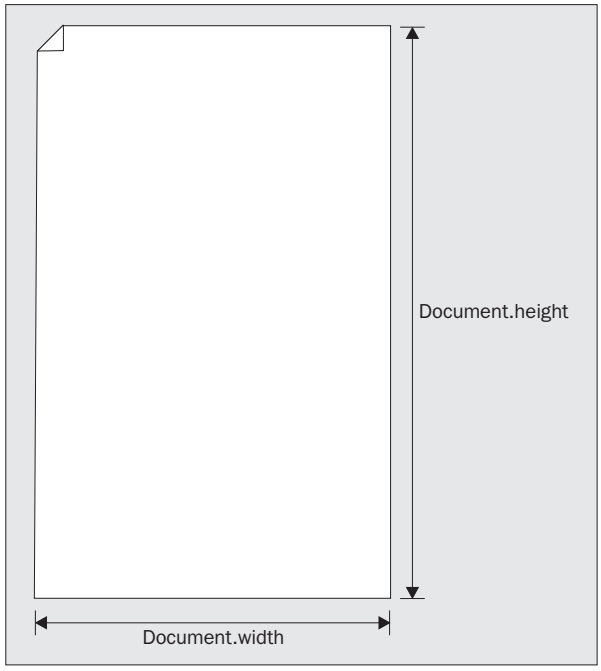
Document.height (Property)

The height of the document.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myDocument*`.body.offsetHeight`
	N	*myDocument*`.height`

The current height of the document measured in pixels. This value constantly changes as the document content is rendered into the page. You can measure this value during document loading and then measure it later to find the page has grown in size.

On MSIE, this property is not supported and you need to do a little work to access the appropriate `Element` object properties of the `body` object.



Warnings:

- This is not supported by MSIE, however you could use the `document.body.offsetHeight` property instead.
- This value can be read by an unsigned script in another window.

See also:	`Document` object, `Document.width`

Document.ids[] (Collection)

Part of the JSS model supported only by Netscape 4.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated	
Property/method value type:	`JSSTags` object	
JavaScript syntax:	N	*myDocument*`.ids`

This yields a `JSSTags` associative array which contains a `style` object for each `Element` object in the page which has an `ID="..."` HTML tag attribute.

The `ID` value is used as an associative array reference to reach the `style` object.

This property does not show up when the document properties are enumerated in a `for( ... in ... )` loop.

Refer to the `JSSTags` topic if you need to operate on these objects.

Warnings:

- This is only supported by Netscape 4 and is part of the JSS support and therefore to be deprecated and should not be used on new projects.
- If you plan to use this, the style settings can only be done during the `<HEAD>` portion of the document and cannot be modified once the document body begins to load.
- This functionality is removed from Netscape 6.0.

See also:	`Document` object, `Document.classes[]`, `ID="..."`, `JSSTags` object

Property attributes:

`ReadOnly`, `DontEnum`.

Document.images[] (Collection)

An array containing a list of all the images in the document.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0
Property/method value type:	`ImageArray` object

JavaScript syntax:	-	*myDocument*.images
HTML syntax:	<IMG>	

Every image in the document corresponds to an element in this array. There is a one for one relationship between these elements and <IMG> tags.

The MSIE and Netscape browsers each maintain an ImageArray object which is just a special case of the Array object. However, although they both store objects that represent images in that array, those image objects are quite different. For a start in Netscape they are of the class "Image" while in MSIE they are of the class "IMG" named after the HTML tag. Perhaps this is fortunate in that you may be able to detect what kind of object you are operating on if you need to perform complex image management activities. This might change however if browsers become more standards compliant in their object class naming, so it may not be true of all versions of all browsers.

The DOM level 1 specification requires that only those images referenced by <IMG> HTML tags should be included in this list.

Warnings:

- Be aware that if you are not using some NAME or ID binding to the <IMG> tags, you may get unexpected results if new images are added and you are accessing elements of this array using numeric index values.

See also:	Document object, Element.all[], Image object, ImageArray object

Property attributes:

ReadOnly

Document.implementation (Property)

A reference to a DOM Implementation object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	Implementation object	
JavaScript syntax:	-	*myDocument*.implementation

This object is used to find out about the DOM implementation in a web browser in the same way that the navigator object tells you about the browser version and its capabilities.

The most useful item belonging to this object is the hasFeature() method. With that you can establsih whether your script is running in an environment that supports the capabilities that you need.

Warnings:

- This may not be implemented in all version of the MSIE browser. It has been reported as being undefined in MSIE 5.0 and 5.5 for Windows.

See also:	`Document` object, `Implementation` object

Property attributes:

`ReadOnly`

Document.lastModified (Property)

The modification date of the document is stored in this property.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myDocument*`.lastModified`

This value is obtained by inspecting the HTTP headers as they arrive in the browser. The value of this header is defined by the web server, which probably used the modification date of the file in the `htdocs` directory that it served the content from.

You can check that this is a valid date by handing it to the `Date.parse()` method as an argument. If `Date.parse()` yields a zero value, then it is invalid unless the file was last modified in January 1970 which is extremely unlikely.

See also:	`Document` object, `Document.fileModifiedDate`, `Document.location`, `Document.referrer`, `Document.title`

Property attributes:

`ReadOnly`

Document.layers[] (Collection)

An array containing a list of layers in the document.

Availability:	JavaScript – 1.2 Netscape – 4.0
Property/method value type:	`LayerArray` object
JavaScript syntax:	N *myDocument*`.layers`

HTML syntax:	<LAYER>

Each item in this array corresponds to a <LAYER> tag in the document. This array also includes layers that are created in Netscape by setting the position attribute of an HTML <DIV> block to absolute.

The layers in this array are ordered according to the order in which they appear in the document. Layers can be accessed associatively if they have been given an ID with the ID="..." or NAME="..." tag attribute. This means you can refer to an element whose ID is set to ABC by its unique name. Either as document.ABC or document layers["ABC"].

Warnings:

- Netscape 6.0 completely removes layer support. If you use layers, your pages will break.
- Be aware in Netscape 4 that if you are not using some NAME or ID binding to the <LAYER> or <DIV> tags, you may get unexpected results if new layers are added and you are accessing elements of this array using numeric index values.
- Unnamed layers will not be added to the array, although the array length will correctly reflect the existence of the layers in its count. The count will be wrong and there will be too few objects in the array when you enumerate them in a loop.

See also:	Document object, ID="...", Layer.layers[], LayerArray object, style.position

Property attributes:

ReadOnly, DontEnum

Document.linkColor (Property)

The color of links that have not yet been visited.

Availability:	DOM level – 0 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0 Deprecated	
Property/method value type:	Color value	
JavaScript syntax:	-	*myDocument*.linkColor
	-	*myDocument*.linkColor = *aColorValue*
HTML syntax:	<BODY LINK="aColorValue">	
Argument list:	*aColorValue*	A hex color value or color name

This value controls the text of active links in the document body. You should use the normal color values to define the required color.

This property is equivalent to the `LINK` attribute of the `<BODY>` HTML tag. This is the color that is used for as yet unvisited links.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse. On the other hand it does work consistently on both MSIE and Netscape where style sheet controls do not.

DOM level 1 deprecates this property in favor of the `BODY.link` property.

Warnings:

- This property can only be changed in the `<HEAD>` section. You cannot modify it after the `<BODY>` has commenced loading.
- On MSIE 5 for Macintosh, the color of the link remains set to its default color until a mouse rolls over it whereupon it changes to the value defined in the script. This is probably a bug.
- Some suggested work arounds include using `document.write()` to output the `<BODY>` tag as the document is loaded. This might work for some applications although I have experienced significant problems with complex pages that are being built with `document.write()` calls. The problems seem to stem from the internal document structure being incomplete and, once the page is loaded, some object references crash the browser. This is particularly problematic with `<OBJECT>` tags in MSIE for Windows and because the `<BODY>` tag is so fundamental to a document, you may want to check that a page composed in this way really does work on all browsers and platforms.

See also:	`BODY.aLink`, `BODY.bgColor`, `BODY.link`, `BODY.text`, `BODY.vLink`, Color names, Color value, `Document` object, `Document.alinkColor`, `Document.bgColor`, `Document.fgColor`, `Document.vlinkColor`

Document.links[] (Collection)

An array of links in the document.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0
Property/method value type:	`LinkArray` object
JavaScript syntax:	- *myDocument*`.links`
HTML syntax:	`<A><AREA>`

This is a collection of `link` objects, each one corresponding to an `anchor` or `area` tag in the document. This array may be identical to the `anchors[]` array if there are no `<AREA>` tags present.

Note that in Netscape , the class type for these link objects is actually `Url` and so they are described in detail at that point in the lexically sorted topics.

Objects are added to this array if they are hyperlinks. Simple named anchors do not qualify for addition to this array although they would be added to the array accessible via the `document.anchors` property. The `anchors`, and `links` arrays allow you to distinguish easily between internal and external HREF values. It is slightly confusing in that all `<A>` tags will be members of the `anchors` array while only externally linking `<A>` tags and all `<AREA>` tags will be in the `links` array.

DOM level 1 requires that this collection should include all objects instantiated by `<AREA>` and `<A>` HTML tags which contain an `HREF="..."` HTML tag attribute.

Warnings:

- Be aware that if you are not using some `NAME` or `ID` binding to the `<A>` and `<AREA>` tags, you may get unexpected results if new links are added and you are accessing elements of this array using numeric index values.
- There are security limitations to what you can do when accessing the links property of a document in another frame. So long as both documents came from the same server, you can access one from the other. Documents from the same domain can access one another as can signed documents according to the security policy in force at run-time.
- It isn't proven conclusively but Netscape 6.0 may have some incipient bugs that prevent `write-access` to properties of objects in this collection. That may be corrected quite quickly as the browser is more widely released.

See also:	`Anchor` object, `Document` object, `Document.anchors[]`, `Element.all[]`, `LinkArray` object, `LinkArray.length`, Security policy, `Url` object, `Url.name`

Property attributes:

`ReadOnly`

Document.location (Property)

This is another name for the `Document.URL` property.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0 Deprecated
Property/method value type:	`Location` object
JavaScript syntax:	- *myDocument*`.location`

This is deprecated now in favor of using the `Document.URL` property to access a `Url` object.

In version 1.0 of JavaScript this was a simple string primitive value that contained the actual location of a window. It was not necessarily the same value as the `Window.location.href` value which always contained the requested location.

As of version 1.1 of JavaScript, this is now a reference to the same object that `Window.location` points at. However at the same time its use became deprecated.

Warnings:

- This `location` property is not the same as the `location` property that belongs to a window. The `document.location` property is a string containing the URL that the document was loaded from. The `window.location` property is a reference to the requested URL encapsulated in an object. Because of this, the `document.location` property is not preferred and you should use the `document.URL` property to avoid confusion.
- This property is supposed to be read-only. However, Netscape 4 allows this property value to be changed. This is not recommended practice and you should use the `Window.location.href` property to change the page in the window.

See also:	`Document` object, `Document.lastModified`, `Document.referrer`, `Document.URL`, `Location` object, `Window.location`

Document.open() (Method)

Open a document body ready for writing.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0

JavaScript syntax:	-	*myDocument*`.open()`
	-	*myDocument*`.open(`*aMimeType*`)`
	-	*myDocument*`.open(`*aMimeType, aReplaceFlag*`)`

Argument list:	*aMimeType*	A text string describing a valid mime types for the document
	aReplaceFlag	A string containing the word "replace"

This opens and prepares the document in readiness for a `document.write()` action.

Although the `document.close()` is required, the `document.open()` may be omitted. Some browsers may be permissive enough to allow the `document.write()` actions to work without a `document.open()` first but occasionally a browser may fail. It is good coding style to observe the discipline of opening a stream before writing to it.

The optional parameter describes the MIME type of the document to be opened. By default this will be "text/html". The "text/plain" MIME type is useful too. You can, in theory, open any kind of document but most binary formats are just not practical.

Refer to the MIME type topic for details of some available MIME type values.

The DOM level 1 specification suggests that this method may be deprecated in the future.

Warnings:

- MSIE does not support the MIME type parameter.

See also:	Debugging – client side, Document object, Document.close(), Document.write(), Document.writeln(), MIME types, Window.open()

Document.parentWindow (Property)

The window object that contains the document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	Window object	
JavaScript syntax:	IE	*myDocument*.parentWindow

This is the parent window in which the document is currently being viewed. This window parent-child hierarchy is the nearest analogy that MSIE has to the Netscape layers facility.

If you write code to manage layers or parent-child windows (such as documents containing in-line frames), you will need to take account of browser differences and code accordingly.

See also:	Document object, Frame object, Window object, Window.document, Window.frame

Property attributes:

ReadOnly

Document.plugins[] (Collection)

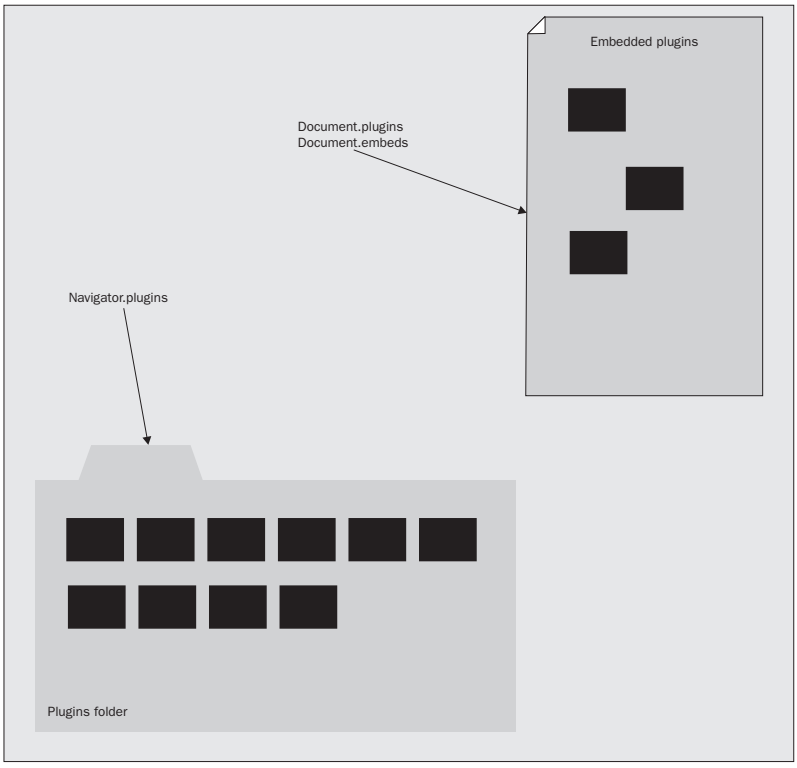
Another (confusing) name for the `document.embeds` property and NOT the `navigator.plugins` property.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0 Deprecated
Property/method value type:	`EmbedArray` object
JavaScript syntax:	- *myDocument*`.plugins`

Each `<EMBED>` tag is represented here by an object. The embeds are encapsulated in the same `JavaObject` objects as are used for the Java applets.

Every embedded plugin will respond to different suites of property and method messages although there may be some similarities between some plugins that serve the same purpose.

Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the EMBED class. In Netscape they are objects commonly referred to as belonging to the `Plugin` class although they are really implemented as `JavaObject` objects. In MSIE, this is an `ActiveX` object.



There is additional confusion in that there is a `plugins[]` array that belongs to the document and another than belongs to the `navigator` object. They both contain collections of objects but of different types. This is further confused by the fact that the `document.plugins[]` array is another name for the `document.embeds[]` array.

Due to this confusing situation, the best recommendation is that we refer to `document.embeds[]` and `navigator.plugins[]` and quietly ignore the `document.plugins[]` array. Furthermore we shall refer to `Plugin` objects as being something the browser can use to play embedded content and `Embed` objects will be an instance of a plugin that is alive and running in a document.

Warnings:

- In MSIE version 4, this returns a `Collection` object instead of an `EmbedArray` object. MSIE 5 is more consistent with Netscape .
- Because of the confusion between embedded and installed plugins, it is highly recommended that you use the `document.embeds` array for scripts that operate on objects instantiated by the `<EMBED>` tag. Then, use the `navigator.plugins` array for operations on the installed and available plugins that may not yet be embedded into a page.

See also:	`Document` object, `Document.embeds[]`, `EmbedArray` object

Property attributes:

`ReadOnly`

Document.protocol (Property)

The protocol that was used when the document was loaded.

Availability:	JScript – 1.0 Internet Explorer – 3.02	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.protocol`

This should yield the value HTTP or FILE or one of the other available protocols according to how the document was accessed. This allows you to build script code that can behave differently according to how the document was loaded.

See also:	`IMG.protocol`, URL

Property attributes:

`ReadOnly`

Document.queryCommandEnabled() (Method)

Part of an MSIE special document command handling mechanism. Indicates if a command is available for a document or text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	IE `myDocument.queryCommandEnabled(aCommandName)`
Argument list:	`aCommandName` An MSIE command name

This method returns a Boolean value that indicates whether the named command is enabled. Many factors can affect the result of this command. It may depend on the ready state of the document or whether a selection is in force.

Refer to the `document.execCommand()` method for a list of the available commands.

Warnings:

- This is only supported by the 32 bit Windows version of MSIE.

See also:	`Document` object, `Document.execCommand()`, `TextRange.queryCommandEnabled()`

Document.queryCommandIndeterm() (Method)

Part of an MSIE special document command handling mechanism. Indicates whether the command is in the indeterminate state.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	IE `myDocument.queryCommandIndeterm(aCommandName)`
Argument list:	`aCommandName` An MSIE command name

If the document is not fully loaded (you can check the `readyState`), or if a command might not be available due to some of its prerequisites not being set (such a selection creating a `TextRange`), this method will return a Boolean `true` value. If it returns a Boolean `false`, then the command may be available as determined by the enabled test.

Warnings:

- This is only supported by the 32 bit Windows version of MSIE.

See also:	`Document` object, `Document.execCommand()`, `TextRange.queryCommandIndeterm()`

Document.queryCommandState() (Method)

Part of an MSIE special document command handling mechanism. The current state of a command for the document or text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive or Null`	
JavaScript syntax:	IE	*myDocument*`.queryCommandState(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

This will return one of the following values:

- Boolean `true` if the command has completed.
- Boolean `false` if it is still in progress.
- Null if the state cannot be determined.

Warnings:

- This is only supported by the 32 bit Windows version of MSIE.

See also:	`Document` object, `Document.execCommand()`, `TextRange.queryCommandState()`

Document.queryCommandSupported() (Method)

Part of an MSIE special document command handling mechanism. Indicates whether the document or text range supports a command.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myDocument*`.queryCommandSupported(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

Some commands are not supported by the `document` object but may be supported by the `TextRange` object.

This method returns a Boolean `true` value if the command is supported by the document object.

Warnings:

- This is only supported by the 32 bit Windows version of MSIE.

See also:	`Document` object, `Document.execCommand()`, `TextRange.queryCommandSupported()`

Document.queryCommandText() (Method)

Part of an MSIE special document command handling mechanism.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.queryCommandText(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

Some commands support the extraction of text from the document or `TextRange`. If the command does support the extraction of text, it will be returned by this method.

Warnings:

- This is only supported by the 32 bit Windows version of MSIE.

See also:	`Document` object, `Document.execCommand()`, `TextRange.queryCommandText()`

Document.queryCommandValue() (Method)

Part of an MSIE special document command handling mechanism. The value of a command for a document or text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.queryCommandValue(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

The value of a command depends on the command itself and what is selected. This method returns a value according to those criteria.

Warnings:

- This is only supported by the 32 bit Windows version of MSIE.

See also:	`Document` object, `Document.execCommand()`, `TextRange.queryCommandValue()`

Document.readyState (Property)

The current downloading status disposition of the document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.readyState`

Sometimes, you can design scripts to execute while the document is downloading. In-line scripts for example. At that time, you may even be able to trigger interval timed deferred executions as well.

If it is important that the document has completed loading, you can check this property for one of the following values:

State	Value
`uninitialized`	The object is first instantiated but has not begun loading.
`loading`	The object has commenced loading.
`loaded`	The object has completed loading.
`interactive`	The object is loaded but not yet closed but is ready to handle interaction.
`complete`	The object body has been closed and the loading is finished.

An object may not need to reflect the complete status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an `OBJECT` object that represents an `<OBJECT>` tag until the `<BODY>` has completed loading. This is because the `ActiveX` object construction requires a complete document body structure to attach itself to.

Every time this `readyState` value changes, it triggers an `onReadyStateChange` event call-back.

See also:	`Document` object, `onReadyStateChange`, `XML` object

Property attributes:

`ReadOnly`

Document.recalc() (Method)

A special MSIE supported method that sends a recalculation event to a document.

Availability:	JScript – 5.0 Internet Explorer – 5.0		
JavaScript syntax:	IE	*myDocument*`.recalc()`	
	IE	*myDocument*`.recalc(`*aFlag*`)`	
Argument list:	*aFlag*		A flag to force the recalculation

All dynamic properties in the document will be recalculated when this method is called.

The flag value is optional and is assumed to be `false` if it is missing. Setting this flag to `false` only recalculates those dynamic properties that have changed.

Setting the flag to the `true` value forces the browser to recalculate all of the dynamic properties regardless of whether they have changed or not.

Document.referrer (Property)

The URL of the document that was displayed when the user clicked on a link to request this document.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myDocument*`.referrer`

The referrer value indicates the page that was being viewed when the user clicked on a link to request this document.

The referrer data is often used in log analysis to trace a click stream that the user traversed your web site with. This property allows you to do similar things at the client end and build all kinds of history trees and back button simulations.

If you are prepared to do a little work to dismantle the referring link, you can detect when someone has a link to one of your internal pages. What you do then is up to you but you could redirect to an error page or simply bump the user to your front page. This would mean that whatever they did to bookmark one of your pages, they would always end up at your front door instead.

Warnings:

- This does not work in MSIE version 3.
- In Netscape, the referrer only contains a value if the document is different. If a document refers to itself in an anchor, clicking on that anchor effectively reloads the document. The referrer will be an empty string. It continues to to be suspect at Netscape 6.0 as well.
- Some log analysis studies suggest that the referrer value is empty in MSIE when documents are loaded into frames.
- Also as a result of studying log results, book-marked links or link values typed into the location box or drag/dropped into the browser will also not yield a meaningful referrer value.

See also:	`Document` object, `Document.lastModified`, `Document.location`, `Document.URL`, `Window.location`

Property attributes:

`ReadOnly`

Document.releaseEvents() (Function)

An alias for the `window.releaseEvents()` method.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myDocument`*`.releaseEvents(`*`anEventMask`*`)`
Argument list:	*`anEventMask`*	A mask defined with the manifest event constants

This is part of the Netscape 4 event management suite which allows events to be routed to handlers other than just the one that defaults to being associated with an event.

The events to be captured are signified by setting bits in a mask.

This method provides a means of indicating which events are no longer needing to be captured by the receiving `Document` object.

The events are specified by using the bitwise `OR` operator to combine the required event mask constants into a mask that defines the events you want to capture. Refer to the Event Type Constants topic for a list of the event mask values.

Since this is only supported by Netscape prior to version 6.0, the functionality is likely to be deprecated when the standards bodies agree on a standard way of handling events. In the meantime, we shall have to implement scripts using this capability if we need to build complex event handling systems that work on legacy browsers. A different script will be required for MSIE although it may be possible for Netscape 6.0 to share the same one.

You may be able to factor your event handler so that you only have to make platform specific event dispatchers, and can call common handling routines that can be shared between MSIE and Netscape.

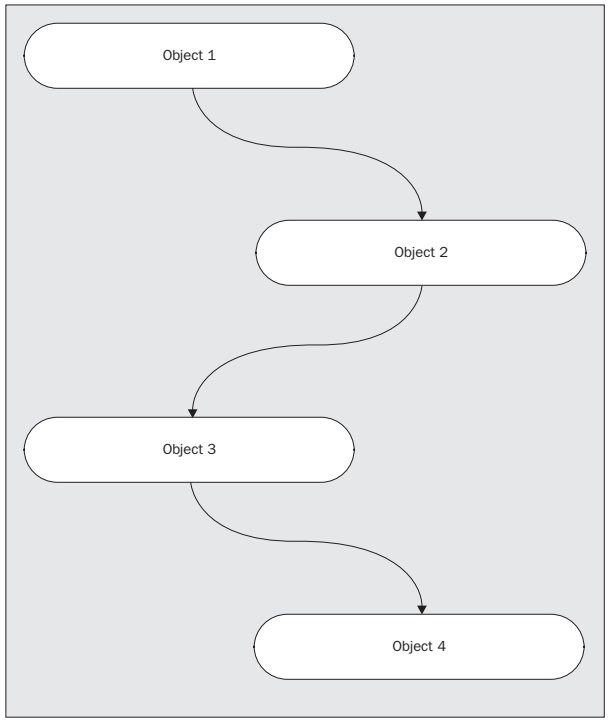
See also:	`captureEvents()`, `Document` object, `Document.captureEvents()`, `Element.onevent`, Event propagation, Event type constants, `Event.modifiers`, `Frame` object, `Layer.captureEvents()`, `Layer.releaseEvents()`, `onMouseMove`, `Window` object, `Window.releaseEvents()`

Document.routeEvent() (Function)

Part of the Netscape event propagation complex.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myDocument`*`.routeEvent(`*`anEvent`*`)`

Argument list:	*anEvent*	An event to be routed



See also:	`Document` object, `Document.handleEvent()`, Window.routeEvent()

Document.scripts[] (Collection)

An array of all the `<SCRIPT>` blocks in a document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`ScriptArray` object	
JavaScript syntax:	IE	*myDocument*`.scripts`
HTML syntax:	`<SCRIPT>`	

This is an array with references to a collection of objects that represent the `<SCRIPT>` blocks in the current document.

Unnamed `<SCRIPT>` blocks are placed into the array with numbered index values. `<SCRIPT>` having an `ID="..."` HTML tag attribute will be added to the array associatively. The `NAME="..."` HTML tag is not reflected in this associative naming scheme and does not work with `<SCRIPT>` blocks.

Warnings:

- In MSIE version 4, this returns a generic `Collection` object instead of a `ScriptArray` object.
- If you interrogate this array in-line while the document is loading, you will not see any script blocks that follow the one containing the script that inspects the array. That suggests this should be used once the document has completed loading.

See also:	`<SCRIPT ID="...">`, `<SCRIPT>`, `Document` object, `ID="..."`, `SCRIPT` object, `ScriptArray` object

Property attributes:

`ReadOnly`.

Document.selection (Property)

The selected text within the object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Selection` object	
JavaScript syntax:	IE	*myDocument*`.selection`

This is the MSIE equivalent functionality to the Netscape `document.getSelection()` method.

Any selected text is returned by a request for this property.

Warnings:

- On Netscape use the `document.getSelection()` method instead.
- On the Macintosh MSIE 5.0 browser, the selection property consistently returns `null` even when there is some text selected. This may be because the Macintosh implementation of MSIE does not support text ranges.

See also:	`Document` object, `Document.getSelection()`, `Selection` object

Property attributes:

`ReadOnly`.

Document.styleSheets[] (Collection)

An array containing a list of style sheets in an MSIE document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`styleSheets` object
JavaScript syntax:	- *myDocument*`.styleSheets`
HTML syntax:	`<LINK>`

This property yields a `styleSheets` array that contains a collection of objects that represent any style sheets currently used by the document. These style sheets are called into the document with the `<LINK>` tag.

The core style sheet model for both MSIE and Netscape is fundamentally the same. However, the style sheet management is done in a completely different way in MSIE and Netscape, prior to version 6.0.

The MSIE technique appears to be far easier to operate on and you can quite easily build algorithmic techniques to locate exactly the `style` object you want to. In Netscape 4, the CSS styling appears to have been added as an interface to an underlying JSS styling model that is far more cumbersome.

As of Netscape 6.0, the JSS support is deprecated and a DOM-based `Style` object like the one in MSIE now provides a more consistent way to operate on styles.

Warnings:

- The entire style management complex is different between the two browsers.

See also:	`Document` object, `rule` object, `StyleSheet` object, `StyleSheetList` object

Property attributes:

`ReadOnly`.

Document.tags[] (Collection)

Part of the JSS model supported only by Netscape 4.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated
Property/method value type:	`JSSTags` object
JavaScript syntax:	N *myDocument*`.tags`

Prior to Netscape 6.0, this was one of the ways into the style sheet manipulation process. It is now deprecated and should not be used in any forward looking new projects unless they really need to operate with legacy browsers. Even then, compared with the MSIE and Netscape 6.0 capabilities, this is quite limited in its scope.

This roughly corresponds to the `Document.styleSheets[]` property in MSIE and is the root of the style control mechanism in Netscape .

Since you cannot enumerate this value and it does not translate easily into a primitive type it is difficult to examine from JavaScript.

Warnings:

- This is not available in MSIE and is part of the Netscape 4 JSS style support and therefore to be deprecated and not used in new projects.
- Although this is supposed to be an associative array, it is really a factory class that manufactures `JSSTag` objects on demand. If you specify a non-existent HTML tag as a property to this object, you should get an error. Unfortunately you just get a new `JSSTag` object.
- This whole JSS style complex is deprecated, and unless you are simply keen to amuse yourself examining it and doing some experiments, it is of little practical use.
- Netscape 6.0 provides a much more capable and portable interface by means of the `Style` object and removes all the JSS functionality.

See also:	`Document` object, `Document.classes[]`, `Document.contextual()`, `JSSTags` object

Property attributes:

`ReadOnly`, `DontEnum`.

Document.title (Property)

The title text for the document.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.all.tags("`*TITLE*`")[0].text`
	-	*myDocument*`.title`
HTML syntax:	`<HEAD><TITLE>aTitleText</TITLE><HEAD>`	
Argument list:	*aTitleText*	Some text in the title heading block

The document title is yielded by this property. The property is not enumerable in the Netscape browser but it is in MSIE. In Netscape this property value is read-only but in MSIE you can modify it whenever you want by assigning a new value to one of the property references that point at the string containing the title text.

This is the text placed inside the `<TITLE>` tags. It exists inside the`<HEAD>` portion of the document, which suggests the document is really rooted at the `<HTML>` tag rather than the `<BODY>` tag. There are minor hierarchical inconsistencies like this.

In the MSIE browser, a special `TITLE` object is also created to help construct a DOM hierarchy inside the browser. The text attribute of that object references the same value as this property.

Warnings:

- For most other objects, this value represents the `TITLE="..."` tag attribute for the tag that constructs the object. Beware when using title attributes that you get the value you really wanted.
- In the MSIE browser, for objects other than the `document` object, the `title` property is used as `ToolTip` text, that automatically appears when the mouse moves over an `Element` object and pauses there.

See also:	`<TITLE>`, `Document` object, `Document.lastModified`, `Element.title`, `HTML` object, `HTML.title`, `LINK.title`, `TITLE` object, `TITLE.text`

Document.uniqueID (Property)

A unique `ID` value for this document.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDocument*`.uniqueID`

Some server side support provides a way to give each document a unique `ID` value. this property reflects such a value for access at the scripting interface.

Document.URL (Property)

This is the actual URL that was loaded for the document.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myDocument*`.URL`

Most of the time, the `document.URL` value will be the same as the `window.location.href` value. However if a server redirect happened, then *`document.URL`* will contain the actual page that was loaded while `window.location.href` will contain the requested URL.

This property should be used in place of the `document.location` property which used to be a string but is now a reference to the same object as the `window.location` property.

See also:	`Document` object, `Document.location`, `Document.referrer`, URL, `Window.location`

Property attributes:

`ReadOnly`.

Document.vlinkColor (Property)

The color of a visited link in the current document.

Availability:	DOM level – 0 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0 Deprecated in DOM level 1	
Property/method value type:	`Color value`	
JavaScript syntax:	-	*`myDocument`*`.vlinkColor`
	-	*`myDocument`*`.vlinkColor = `*`aColorValue`*
HTML syntax:	`<BODY VLINK="aColorValue">`	
Argument list:	*`aColorValue`*	A hex color value or color name

This value controls the text of visited links in the document body. You should use the normal color values to define the required color.

This corresponds to the `VLINK` attribute in the `<BODY>` tag.

Now that the style control facilities are more sophisticated, this tag attribute is likely to fall into disuse. On the other hand it does work consistently on both MSIE and Netscape where style sheet controls do not.

DOM level 1 deprecates the use of this property in favor of the `BODY.vLink` property.

Warnings:

- This property can only be changed in the `<HEAD>` section. You cannot modify it after the `<BODY>` has commenced loading.

See also:	`BODY.aLink`, `BODY.link`, `BODY.text`, `BODY.vLink`, Color names, Color value, `Document` object, `Document.alinkColor`, `Document.bgColor`, `Document.fgColor`, `Document.linkColor`

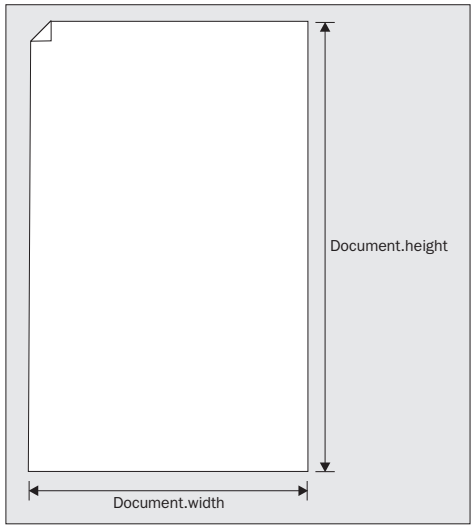
Document.width (Property)

The width of the document.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myDocument*`.body.offsetWidth`
	N	*myDocument*`.width`

The current width of the document measured in pixels. This value constantly changes as the document content is rendered into the page. You can measure this value during document loading and then measure it later to find the page has grown in size.

On MSIE, this property is not supported and you need to do a little work to access the appropriate `Element` object properties of the body object.



Warnings:

- This is not supported by MSIE, however you could use the `document.body.offsetWidth` property instead.
- This value can be read by an unsigned script in another window.

See also:	`Document` object, `Document.height`

Document.write() (Method)

A method for writing HTML into the document body.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
JavaScript syntax:	-	*myDocument*.write(*anArgument*)
	-	*myDocument*.write(*anArgument, ...*)
Argument list:	*anArgument*	A value to be written out to the document

In a client-side script, you would use `document.write()` to generate some HTML.

This is a host method. It belongs to the `document` object. Its argument values are converted to a string value and are then appended to the content of the document window and then interpreted as HTML.

This is the primary means of generating HTML as a document is parsed and the scripts are executed. Everything that is output by the `document.write()` method is streamed into the page at the execution point where it is called. That means if you place a `<SCRIPT>` block in the middle of the page and it has a `document.write()` in its global code (that is not inside a function declaration), the output will be inserted into the document in place of the `<SCRIPT>` block.

This is a useful technique for creating dynamically changing pages where the dynamism happens at the client end.

Although writing to the current document in an event handler will destroy the document, you can perform `document.write()` actions in other windows to replace their document content. You will need to invoke the `document.write()` method belonging to the target window or frame you want the write to happen in. Here is a `document.write()` that is targeted at a frame somewhere in a frame-set:

```
top.frames[4].document.write("Some target content");
```

When writing across frames like this, you should call the `document.open()` method before the `document.write()` and then call `document.close()` afterwards. The animated browser loading icon will continue to revolve until you close the target document.

The `document.write()` method takes a variable number of arguments which it will concatenate in the output. You can do the concatenation manually but all that is needed is to comma separate the individual items for the `document.write()` to perform this step automatically.

The DOM level 1 specification suggests that this method may be deprecated in the future.

Warnings:

- Beware if you use `document.write()` at any time other than during the document parsing. If you use it in an event handler, you will overwrite the entire documents. Once you have overwritten the document, even the event handler has gone and there is no trace of the original document so when you subsequently refresh again, you may get a blank page.

- Destroying the document content with a `document.write()` will crash Netscape 2 browsers. It may lead to unexpected behavior in Netscape 3 browsers.
- Beware that if you were to use a `with` statement to add the document object to the scope chain, your script would look very much like the server side scripts which use the `write()` method on its own.
- If the output that you write does not immediately appear, it may have been buffered by the browser. There is no way to flush this buffer out to the screen and keep the document open. The only way to flush the buffer is to call the `document.close()` method. However any subsequent `document.write()` will implicitly call `document.open()` and clear the page as it starts a new empty document.
- If you need to change the HTML at that time, you will need to resort to the following dynamic HTML techniques:
  - `<DIV>` blocks
  - `<SPAN>` blocks
  - `innerHTML` and `innerText` properties of elements
  - `createElement()` and its related DOM support.

See also:	Comma operator (,), `Document` object, `Document.close()`, `Document.open()`, Input-output, `response.write()`

Document.writeln() (Method)

A method for writing HTML into the document body.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
JavaScript syntax:	-	*myDocument*`.writeln(`*anArgument*`)`
	-	*myDocument*`.writeln(`*anArgument, ...*`)`
Argument list:	*anArgument*	A value to be written out to the document

The `document.writeln()` method is very similar to the `document.write()` method. The difference is that `document.writeln()` will place a carriage return after the written value.

This is of little consequence when writing HTML because the browser ignores any line breaks in the HTML.

However, it can be useful when writing other kinds of output such as plain text for example.

The DOM level 1 specification suggests that this method may be deprecated in the future.

See also:	Comma operator (,), `Document` object, `Document.close()`, `Document.open()`, Input-output, `response.write()`

DocumentEvent (Object/DOM)

An interface that extends the `Document` object to support a DOM compliant event structure.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myDocumentEvent = myDocument*
Object methods:	`createEvent()`	

This is not an object class that exists on its own. Rather it is an extension to the underlying `Document` object. It could be considered to be a sub-class of `Document` that inherits all the properties and methods of the `Document` class. Objects of this type are likely to report that they belong to the `Document` class rather than the `DocumentEvent` class although that may be implementation dependent.

The syntax listing shows that a `DocumentEvent` object is equal to a `Document` object although as is the case with Document, you cannot be certain which one of several possible documents you may have been passed if several windows, frames or layers are in use at once.

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`createEvent()`	1.5 +	-	6.0 +	-	-	2 +	-

DocumentEvent.createEvent() (Method)

A method to create a new `event` object ready to be dispatched to an `EventTarget`.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`Event` object	
JavaScript syntax:	N	`myDocumentEvent.createEvent(aType)`
Argument list:	*aType*	A string containing an event type

The only argument to this method contains the event type value. This is discussed fully in the `Event.type` topic elsewhere. The method returns a freshly manufactured but uninitialized `Event` object.

The DOM specification describes how the event should be initialized by an appropriate initialization method.

During the execution of this method, it is possible to raise a DOM exception. The only one enumerated in DOM level 2 is the `NOT_SUPPORTED_ERR` value.

The following event types are defined in DOM level 2:

Value	Event object type
`HTMLEvents`	HTML `Element` objects
`MouseEvents`	`MouseEvent` object
`MutationEvents`	`MutationEvent` object
`UIEvents`	`UIEvent` object

See also:	`Event.type`, `UIEvent.initUIEvent()`

DocumentFragment object (Object/DOM)

The DOM specification calls this a lightweight or minimal document object. It can be used as a temporary store for a part of the document hierarchy.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0
Inherits from:	`Node` object
JavaScript syntax:	N *myDocumentFragment* = *myDocument*`.createDocumentFragment()`

See also:	`Document.createDocumentFragment()`, `Node` object

Inheritance chain:

`Node` object

DocumentStyle object (Object/DOM)

Added at DOM level 2 to support document related stylesheets.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0
JavaScript syntax:	N *myDocumetStyle* = new DocumentStyle()

This is a major upgrade in progress. At present the DOM level 2 standards run to several hundred pages and are extremely powerful.

The event model is supported by the Netscape 6.0 browser. Some (probably most) of the style model is supported by MSIE and Netscape 6.0 but there are some shortcomings. For instance, MSIE incorrectly names some properties and object types and it introduces some non-compliant extra methods and properties which are not portable.

The document styling interface is likely to be an area of major amounts of work on the next round of browser upgrades.

DOM mandates that this object should have a single property:

- `styleSheets`

Note that MSIE takes that property name and uses it as an object type. DOM mandates that the property should point at a `StyleSheetList` object and not a `styleSheets` object.

DocumentType object (Object/DOM)

This is implemented in MSIE as a Doctype object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Inherits from:	`Node` object	
JavaScript syntax:	-	`myDocumentType = new DocumentType()`

Inheritance chain:

`Node` object

Refer to:

`Doctype` object

DOM (Standard)

A standardized model of a document built with objects.

The DOM standard is concerned with mapping document components to an object model which reflects the values defined in HTML tag attributes. These are visible to the script writer as object properties. In case of conflicts with reserved words and object names in the JavaScript context, any conflicting names will have the"`html`" prefix. An example is the "`for`" attribute which becomes the "`htmlFor`" property. The HTML tag that instantiated an object is reflected in the `tagName` property of each element object. All document elements derived from HTML tags are sub-classed from the `Element` class.

The W3C Document Object Model standard is being reviewed and updated to enhance the support of the document in the browser. There are several levels to this standard:

- Level 0 – The more or less de-facto situation with the version 4 browsers.
- Level 1 – Text, elements and attributes of HTML and XML.
- Level 2 – Views, traversals, events and stylesheets standardized.
- Level 3 – More work on events and the content model introduced.
- Future – Potential standardization of the security model and standardization of the context and environment in which the document exists.

Thankfully now that browsers are converging on the same standards the amount of duplicated effort will diminish over time as the older browsers are replaced. Netscape 6.0 is just beginning to ship. Beta versions of MSIE version 6.0 are imminent and Opera version 5 is likely to be similarly capable as regards DOM compliance. DOM level 1 seems to be roughly where we are at present.

Browser manufacturers make grand claims to be ECMAScript compliant as well as DOM compliant. This claim is somewhat suspect. Providing objects with the functionality of DOM specified classes but having different class names does not completely satisfy the requirements for DOM compliance. We may ultimately end up with objects being mirrored into duplicate instances under different class names to satisfy DOM class naming and to preserve legacy support.

MSIE at version 5 supports a DOM like object model with Microsoft specific class names. Netscape 6 supports a highly DOM compliant object model with correct object class names. When testing the PR3 beta version of Netscape 6.0, it looked like several HTML tags instantiated objects of the same class when they should have been different but this may have gone away in the final release.

As browser manufacturers support more standardized interfaces, we may be better off in some areas but are also likely to be inconsistent between the browsers in some new areas.

As the new levels are introduced and add new modules, they often extend the interfaces of existing classes. The DOM standard accomplishes this by defining new classes as if they were a sub-class of the object they extend. This provides some opportunity for implementors to name object classes incorrectly in early version of their DOM support. For example, event handling extends the `Document` object to allow it to create new events. This would really be an extension of the `Document` object and would likely not be implemented by sub-classing `Document` to create a `DocumentEvent` object. Were that the way the implentor had chosen, we would have a DOM hierarchy model that had been structurally altered by the insertion of a new sub-class between Document and HTMLDocument and we already have enough confusion between those two object classes across browsers.

Warnings:

- The support for DOM in Netscape 4 is so vestigial that it cannot really be called a DOM implementation at all. This is corrected in Netscape 6.0 which supports DOM level 1 and additionally supports some DOM level 2 features to do with Event handling.

See also:	`CLASS="..."`, Document, Document component, Dynamic HTML, `Element` object, `MutationEvent` object, `NamedNodeMap` object, Overview

Web-references:

http://www.w3c.org/DOM/

DOM - Level 0 (Standard)

The initial collation of document objects and properties from the de-facto HTML & JavaScript implementations.

The level 0 version of the DOM standard was compiled from the available functionality of the Netscape 3.0 and MSIE version 3.0 functionality. The principal input was derived from the HTML support provided by the browsers. As it stands, some attributes and methods are provided to support backwards compatibility with these older browsers. Those items will likely become deprecated features and will in due course be withdrawn.

DOM - Level 1 (Standard)

A standardized model of a document built with objects.

The level 1 version of the DOM standard defines the complete content structure of the document.

This is implemented in the standard in two parts:

- DOM core functionality
- HTML functionality

The core functionality should apply to documents of a variety of types while the HTML functionality should only be supported in implementations that use HTML marked up documents.

In a fully compliant browser, you can expect to access all of the attributes, names and values of each and every tag. There should be a high degree of consistency between HTML tag attribute sets and object properties and methods.

The document should then be represented as a tree of objects starting at the outermost `<HTML>` tag and all subsequent items being contained in a logical tree structured parent and child arrangement.

With the consistent compliance between HTML attributes and JavaScript object properties, there may still be a few catch-outs. For example, the namespaces may collide. An HTML attribute may correspond to a JavaScript reserved word and therefore it should not be used as a property or method name since they exist in the identifier namespace. Identifiers and JavaScript reserved words must not collide.

To work around this, some prefix will be added to the property names.

The Level 1 DOM support should also include some API support to locate objects by name without necessarily walking the document tree.

Warnings:

- The level 1 DOM specification is similar to but may be somewhat incompatible with the MSIE version 4 DOM implementation. Netscape 6.0 provides the most accurate and compliant implementation of DOM level 1 at the time of writing.

See also:	`NamedNodeMap` object

DOM - Level 2 (Standard)

A standardized model of a document built with objects.

The level 2 DOM is an enhancement on the level 1 support. It introduces additional properties and styles for `document` objects.

The standard is composed of the following modules:

- DOM core functionality
- HTML functionality
- Event handling

- Style specification
- Document views
- Traversal and range specification

The DOM level specification is now a 500 page document that maps the HTML specification to the ECMAScript and Java language bindings. Many additional objects are provided to assist in navigating the DOM hierarchy.

In general, the MSIE version 5 browser on Windows and Macintosh is the best and most complete implementation of DOM released for general use so far.

Netscape 5 never saw the light of day, and with Netscape 6, the development teams have a stated goal of going very aggressively for DOM standardization. Running the same inspection scripts on MSIE 5 and Netscape 6 reveals that the DOM implementations are structurally similar but that the DOM compliance is more robust in Netscape . This is because MSIE does not use the standardized class names `fpr` objects although the entity relations are more or less correct.

The differences between level 1 and level 2 basic support for DOM are as follows:

- Some new methods added to the existing DOM interfaces and exceptions. This extends to about 30 minor changes.
- Many wholly new interfaces are added. These provide access to the HTML implementation, views on the document, additional `StyleSheet` support, new CSS support, `Event` handling support, traversals and ranges.

See also:	ECMAScript, `MutationEvent` object

DOM - Level 3 (Standard)

An improved model of the document object structure.

The level 3 DOM is an enhancement on the level 2 support. It introduces additional properties and interfaces for `document` objects.

The standard is composed of the following modules:

- DOM core functionality
- HTML functionality
- Event handling
- Style specification
- Document views
- Traversal and range specification
- Content model and archiving

The DOM level specification has grown again although not as much as it did at the level 2 version. Some changes to the `Core` and `Events` modules have taken place and a new module has been introduced to support content models and the ability to export and import documents between implementations.

Level 3 standardization is still very much a work in progress and as a whole the DOM standardization process is somewhat held back by work in other groups to do with XML and internationalization among other things. Progress is still being made and although there is much to do, many aspects of the DOM standard are now reasonably well defined and not likely to change.

It still remains for the browser manufacturers to catch up with the standard and in particular to use the same naming conventions for object classes instead of continually inventing their own.

See also:	ECMAScript

DOM Events (Standard)

A new modular part of the DOM standard introduced at level 2 and implemented in Netscape 6.

The DOM level 2 event model has been designed to be platform and language neutral. The standard describes bindings for Java and ECMAScript and it is applicable to other environments too.

The foundation on which DOM Events are constructed requires that DOM Core and DOM views are available too.

There are three main types of events:

- User interface related events such as those triggered by an input device. A key press or mouse click is an example.
- Logical events that may have been triggered via the UI but are more abstract. A focus change or element notification event is an example.
- Mutation events which are caused by some action that modified the structure of the document.

Events are captured in several ways depending on the implementation and context. For example, server-side events would be captured in a different way to those in a web browser.

In a structured document, a target element that receives an event may handle it or may choose to pass it upwards through the document hierarchy to its parent node. This is called event bubbling and was first provided in a web browser by MSIE. As of version 6, the same event model is used in Netscape which attempts to implement the entire DOM level 2 event capabilities as described in the standard.

Events can be cancelled. This can prevent them from bubbling upwards or from exercising some default behavior.

As an alternative to implementing an event bubbling technique, the DOM level 2 event model also provides for an event capturing approach where the highest ancestor object that registers a listener will receive the event before the target object. This traverses the document hierarchy from top to bottom and is the opposite of the bubbling technique which traverses from bottom to top.

`EventTarget` objects are not really objects in their own right although it is convenient to describe them thus in the DOM specification. An `EventTarget` is actually one of the already existing classes but when the DOM level 2 event model is implemented, the `element` objects that can react to events become `EventTargets` by inheriting the properties and methods of the `EventTarget` class as well as any others they inherit from their superclass.

Events are handled by registering `EventListener` functions. These are registered by the potential `EventTargets`. An `EventTarget` is simply an HTML instantiated object, or one that has been manufactured by script and placed into the display for the user to interact with.

See also:	`Element` object, Event, Event management, Event model, `MutationEvent` object

Domain error (Definition)

An error in computation that would normally crash a compiled program.

Refer to:

Minima-maxima

DOMImplementation object (Object/DOM)

MSIE implements this class as the `Implementation` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
JavaScript syntax:	- *myDOMImplementation* = new DOMImplementation - *myDOMImplementation* = myDocument.implementation

Refer to:

Implementation object

DontDelete (Property attribute)

An internal property attribute that prevents a property from being deleted.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

This attribute is set internally on properties by the host environment that the interpreter implementation is running in. When this attribute is set on a property, that property cannot be deleted by the script writer. This attribute is not normally exposed to the script level code.

See also:	DontEnumerate, ReadOnly

Cross-references:

ECMA 262 edition 2 – section – 11.4.1

DontEnumerate (Property attribute)

An internal property attribute that prevents a property from being enumerated.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

This attribute is set internally on properties by the host environment that the interpreter implementation is running in. When this attribute is set on a property, that property cannot be enumerated by the script writer. This attribute is not normally exposed to the script level code.

See also:	DontDelete, `for( ... in ... ) ...`, ReadOnly

Cross-references:

ECMA 262 edition 2 – section – 12.6.3

ECMA 262 edition 3 – section – 12.6.4

double (Reserved word)

Reserved for future language enhancements.

Providing `double` as a reserved keyword suggests stronger data typing may be available in later versions of the ECMA standard.

This keyword also represents a Java data type and the `double` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	float, Integer, `java.lang.Double`, LiveConnect, long, Reserved word, short

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Double-precision (Definition)

A type of number value.

Double precision numbers use twice the number of bits to represent the value and this increases the range of values that can be resolved by the numeric computations.

See also:	Divide (/), JellyScript

Drive object (Object/JScript)

A special JScript object to represent a disk drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0
JavaScript syntax:	IE `myDrive = myFileSystem.GetDrive()`
Object properties:	`AvailableSpace, DriveLetter, DriveType, FileSystem, FreeSpace, IsReady, Path, RootFolder, SerialNumber, ShareName, TotalSize, VolumeName`

This is an object that represents the disk drive that a `file` object is stored on in the Windows environment. This object is accessed via the `GetDrive()` method belonging to the `FileSystem` object.

See also:	`FileSystem.GetDrive(),Folder.Drive`

Property	JavaScript	JScript	N	IE	Opera	Notes
`AvailableSpace`	-	3.0 +	-	4.0 +	-	-
`DriveLetter`	-	3.0 +	-	4.0 +	-	-
`DriveType`	-	3.0 +	-	4.0 +	-	-
`FileSystem`	-	3.0 +	-	4.0 +	-	-
`FreeSpace`	-	3.0 +	-	4.0 +	-	-
`IsReady`	-	3.0 +	-	4.0 +	-	-
`Path`	-	3.0 +	-	4.0 +	-	-
`RootFolder`	-	3.0 +	-	4.0 +	-	-
`SerialNumber`	-	3.0 +	-	4.0 +	-	-
`ShareName`	-	3.0 +	-	4.0 +	-	-
`TotalSize`	-	3.0 +	-	4.0 +	-	-
`VolumeName`	-	3.0 +	-	4.0 +	-	-

Drive.AvailableSpace (Property)

The amount of free space on the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Number primitive`
JavaScript syntax:	IE `myDrive.AvailableSpace`

The amount of total capacity on the drive may be useful to a server-side script developer who needs to store some information during a session. Refer to the `FreeSpace` property for details of the remaining space available on the drive.

See also:	`Drive.FreeSpace,Drive.TotalSize`

Drive.DriveLetter (Property)

The name of the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDrive*`.DriveLetter`

The abbreviated letter name of the drive in the Windows file system hierarchy.

If the `Drive` object was supported on file-systems other than Windows, the behavior of this property would be unclear, as only Windows names its drives by letters.

Drive.DriveType (Property)

The kind of disk media in the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myDrive*`.DriveType`

This returns a string describing the kind of media that the drive supports. These drive types are currently supported:

Type	Description
0	Undefined
1	Removable volume
2	Fixed disk
3	Network mounted drive
4	CD-ROM
5	RAM disk in memory

If you know the type of drive being used, you can write scripts in a more informed manner. If you are aware that the drive may be volatile, then you can make sure things are backed up, and you can also establish whether the volume is updateable from its type.

Drive.FileSystem (Property)

The kind of file system on the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDrive`*`.FileSystem`

A disk drive is mapped via a file-system. This property returns a reference to an object that encapsulates the file system on the drive.

See also:	`FileSystem` object

Drive.FreeSpace (Property)

The amount of free space left on the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myDrive`*`.FreeSpace`

The amount of spare capacity on the drive may be useful to a server-side script developer who needs to store some information during a session.

See also:	`Drive.AvailableSpace`, `Drive.TotalSize`

Drive.IsReady (Property)

Whether the drive is ready to be used.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myDrive`*`.IsReady`

The behavior of this property will depend on the hardware you have in your system. If you use a hard disk drive, it is likely to always yield `true` for this property. Other drive types may need to be spun up or mounted. They may yield a `false` value. Some may yield a `false` value that changes to true after a short while.

Drive.Path (Property)

The path for the specified drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDrive`*`.Path`

If the drive is mounted into a file system at some location other than the root, this is the path to reach its root directory.

Drive.RootFolder (Property)

A `folder` object that represents the root folder for the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDrive`*`.RootFolder`

Drives have a root folder. This property yields the path to reach the root folder for the drive that this object encapsulates.

Drive.SerialNumber (Property)

The serial number of the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDrive`*`.SerialNumber`

Not all drives will provide a serial number. Some removable cartridge drives support this as a means of identifying cartridges when they are used in multiple drives or systems.

Drive.ShareName (Property)

If the drive is shared, then this is its shared name.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDrive`*`.ShareName`

If a disk drive is shared for use by other users on a network, this property reflects the name it was given when it was set up for sharing.

Drive.TotalSize (Property)

The total space available on the disk if it were empty.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myDrive`*`.TotalSize`

This is the total space available on the drive. There appears to be some redundancy here given that we also have `AvailableSpace` and `FreeSpace` properties.

See also:	`Drive.AvailableSpace`, `Drive.FreeSpace`

Drive.VolumeName (Property)

Access to the volume name of the drive.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myDrive`*`.VolumeName`

If the disk drive supports volume naming, this may be an alternative way to identify a disk drive as opposed to the `DriveLetter` property value.

Drives object (Object/JScript)

A collection of drives belonging to a file system.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myDrives* = myFileSystem.Drives
Object properties:	count	

The `Drives` object is a collection that needs to be used with an enumerator to access the individual drive items.

Example code:

```
// Instantiate a file system object
myFileSystem = new ActiveXObject("Scripting.FileSystemObject");

// Create an enumerator
myEnum = new Enumerator(myFileSystem.Drives);

// Traverse the Drives collection via the enumerator
for(; !myEnum.atEnd(); myEnum.moveNext())
{
  processDrive(myEnum.item());
}

//A function to do something with each disk drive
function processDrive(aDrive)
```

See also:	FileSystem.Drives[]

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
count	-	3.0 +	-	4.0 +	-	-	-	-	-	-	-

DropShadow() (Filter/visual)

A visual filter for creating drop shadows.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Filter – `DropShadow()`

DT object (Object/HTML)

An object that represents the content of a <DT> tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myDT* = *myDocument*.all.*anElementID*
	IE	*myDT* = *myDocument*.all.tags("DT")[*anIndex*]
	IE	*myDT* = *myDocument*.all[*aName*]
	-	*myDT* = *myDocument*.getElementById(*anElementID*)
	-	*myDT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myDT* = *myDocument*.getElementsByTagName("DT")[*anIndex*]
HTML syntax:	<DT> ... </DT>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	noWrap	
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

This object represents the definition term of an item in a definition list. It corresponds to a related definition contained in a `DD` object. `DT` and `DD` objects are paired up and maintained together as a member of the `DL` collection.

The <DT> tag is a block-level tag. That means that it forces a line break before and after itself unless its enclosing <DL> tag uses its compact property.

See also:	`DD` object, `DL` object, `Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
noWrap	-	3.0 +	-	4.0 +	-	-	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
OnFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

DT.noWrap (Property)

Controls the wrapping of text within a `<DT>` block.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myDT*`.noWrap`

This is a Boolean value that controls whether the textual content is wrapped at the right hand window border or not.

If the value `false` is assigned to this property, then words will wrap as the page is drawn. This is good and is the way you would expect a browser to behave. The text will flow according to the space available.

If the value `true` is assigned to this property, the line of text will continue to the right until a `<BR>` or other block level tag is encountered. This will force the horizontal width of the page to extremely large and the user will need to scroll furiously to be able to see the text and then scroll back again for the start of the next line.

Warnings:

- Only use this if you plan to place line breaks at frequent intervals yourself and really do need to control the line breaks manually.

DVB-MHP (Standard)

Digital Video Broadcasting – Multimedia Home Platform.

An emerging Digital TV platform standard, which may achieve some dominance in the future. Of interest to JavaScript users because it is being developed by Sun Microsystems as a platform in which a web browser could be hosted.

If that is the case, there may, by implication, be a JavaScript interpreter available for use.

See also:	Interpret

Dynamic HTML (Definition)

A fourth generation browser technology for dynamically altering the document that describes a web page.

Dynamic HTML is a term that collectively describes the DOM (Document Object Model) and the script activated mechanisms for changing the content of a web page either interactively or not. The content of the page can be animated or modified as a result of the user moving the mouse or clicking on items in the page. The event model is also a contributing factor as is Style sheet support for abstraction of content and appearance by means of CSS (Cascading Style Sheets) and the absolute positioning of HTML elements.

Really the term Dynamic HTML should be deprecated in favor of "DOM scripting" and "CSS scripting".

See also:	CSS, DHTML, DOM, Event model

Dynamic positioning (Definition)

Cascading style properties for positioning objects within the page.

Dynamic positioning in MSIE is accomplished with the `style` object that belongs to each `Element` object in the MSIE DOM. This means that every tag pretty much corresponds to an object in the MSIE DOM. This gives a very fine-grain approach to style and position attributes.

Netscape prior to version 6.0 had to accomplish dynamic positioning by means of layers. This is a bit more complicated than the CSS positioning now available in the new version.

Warnings:

- The JavaScript API that Netscape and MSIE provide to support dynamic positioning differ from one another.

See also:	CSS level 2, CSS-P, style.position

E-mail containing JavaScript (Advice)

You can embed JavaScript into e-mail messages composed using HTML.

When you compose and send an e-mail message, you may use HTML as a way to improve the presentation. This means you can include some JavaScript to be executed in the client mail reader application.

Not all mail clients can support HTML, let alone JavaScript. However, if your recipient does, then you can do some creative things to generate auto reply messages (security permitting). You can also do animation, play audio, present a form for a survey etc. You simply construct your HTML document in the normal way.

Server-generated mail-outs could use this technique to determine how well targeted the mailing is. Perhaps you could put a subscribe/unsubscribe button on the form and tie that to a user ID. That would get round one of the major difficulties of unsubscribing people, which is to do with the mutations that mail addresses undergo. Periodically, a mail server may move or a domain name may change, and this means that unsubscribing from the new address does not locate the subscription record that was made under an old address. Passing an ID back and forth solves this major headache.

Warnings:

- There are significant security and virus related risks with JavaScript enabled e-mail. The possibilities are so catastrophic that the best recommendation is to deactivate JavaScript and Java in any mail reading client application.
- Just because something is possible does not mean it is advisable or good to do.
- On the other hand, within the confines of a closely controlled Intranet or workgroup, this could find many useful applications. Just so long as you know where the mails came from and you can absolutely trust that they have not been compromised.
- Personally, I'd recommend that you turn it off. I've not yet been convinced of the need to support HTML in e-mail correspondence other than for demonstrating how hip and trendy I can be.

See also:	News posts containing JavaScript

Cross-references:

Wrox *Instant JavaScript* –page –60

ECMA (Standard)

An international standards organization.

The European Computer Manufacturers Association (ECMA) has be developing standards for computing systems for many years.

You can obtain printed copies of their standards and in most cases you can download an electronic copy for your own use.

You should explore the ECMA web site for details of their activities and standards if you are developing compliant implementations or scripts.

See also:	Compliance, Conformance, ECMAScript, Implementation, Opera, Overview, Topic classification

Web-references:

ftp://ftp.ecma.ch/mailto://documents@ecma.ch/http://www.ecma.ch/

ECMAScript (Background)

An international standard that describes JavaScript.

ECMAScript is an object oriented programming language for performing computations and manipulating computational objects within a host environment.

It defines the central or core capabilities of the language and does not define any of the host defined capabilities.

As defined in the standard, ECMAScript is not intended to be self-sufficient, but should provide core functionality on top of which host objects need to be added.

Scripting languages are intended to be used by both experienced programmers and non-programmers. This means a scripting language will tend to be less formal than a compiled language and will relax the rules a little. Basically you can get away with more things than a compiler would permit.

Scripting languages are generally used to automate existing capabilities of a hosting environment. Those facilities may be already accessible under manual operation via a Graphical User Interface.

ECMAScript was originally designed to be a web scripting language to provide facilities to add dynamics to client-side browser displayed web pages.

Some of the facilities provided by ECMAScript interpreters are similar to those available in other languages such as Java, Perl and C.

Here is an extract from the ECMA second edition standard:

"This ECMA Standard is based on several originating technologies, the most well-known being JavaScript and JScript. The language was invented by Brendan Eich at Netscape and first appeared in the Netscape Navigator version 2.0 browser. It has appeared in all subsequent Netscape Navigator browsers and in all browsers from Microsoft starting with MSIE version 3.0."

Warnings:

- The browser manufacturers go to great lengths to stress how well they comply with a standard. Microsoft and Netscape both claim compatibility with ECMAScript.
- It is not always clear which revision of ECMAScript they are claiming compliance with. It is hoped that they mean edition 3 of the standard. However if they don't make this clear, they may only be edition 2 compliant.
- If all the browsers you are using were 100% compliant with ECMAScript edition 3, would this then mean that you could write one version of your script and deploy it everywhere?
- Unfortunately not. ECMAScript only defines the core language. It allows for the browser manufacturers adding host specific features in a compliant fashion, but it does not specify what they should be. Because the DOM implementations are so different, claiming ECMAScript compliance is almost meaningless in terms of whether you can really build portable scripts to run in web browsers.
- That is not to diminish the importance of ECMAScript at all. It is the foundation on which every JavaScript interpreter should be based – without question. However, for your script to be truly portable, every other facet of the implementation must also conform strictly to the same standard.
- At least we could expect browser manufacturers to comply with ECMAScript and the W3C DOM definition. That still leaves many holes in security standardization and event handling. Not to mention the new directions regarding mobile devices and TV set-top boxes, which will all lead to diverging implementations.
- The event handling is being rigorously worked out as part of the DOM level 2 and level 3 standardization work.

See also:	Compliance, Conformance, DOM – Level 2, DOM – Level 3, ECMA, ECMAScript – edition 2, ECMAScript – edition 3, ECMAScript version, Glue code, Host features, Implementation, JavaScript language, Native feature, Opera, Overview

Cross-references:

ECMA 262 edition 2 – section – 4

ECMA 262 edition 3 – section – 4

Wrox *Instant JavaScript* – page – 12

ECMAScript – edition 2 (Standard)

Specific conformance for ECMAScript at its second edition.

The character encoding support described in the second edition of the standard is very specific:

"A conforming implementation of the International standard shall interpret characters in conformance with the Unicode Standard, Version 2.0, and ISO/IEC 10646-1 with UCS-2 as the adopted encoding form, implementation level 3. If the adopted ISO/IEC 10646-1 subset is not otherwise specified, it is presumed to be the BMP subset, collection 300."

Other than typographical and editorial changes, little was added between the first and second editions of the ECMA standard. That means this edition becomes the foundation specification for the core language.

The principle features of this edition include:

- Arithmetic, string and logical operators
- Global and local variables managed via scope chains
- Core data types and objects (`Number`, `String`, `Boolean`)
- Core object types (`Date` and `Array`)
- Core numeric library support by means of the `Math` object
- Language flow control and keywords
- Object instantiation by means of constructors
- Inheritance supported via prototype chains

See also:	Compliance, Conformance, ECMAScript, Implementation

Cross-references:

ISO/IEC 10646-1 Information Technology – Universal Multiple-Octet Coded Character Set (UCS), plus its amendments and technical corrigenda.

The Unicode Standard, Version 2.0, Addison-Wesley Publishing Co. ISBN: 0-201-48345-9

ECMAScript – edition 3 (Standard)

Specific conformance for ECMAScript at its third edition.

The character encoding support described in the third edition of the standard is somewhat more advanced than that in the second edition:

"A conforming implementation of this International standard shall interpret characters in conformance with the Unicode Standard, Version 2.1 or later, and ISO/IEC 10646-1 with either UCS-2 or UTF-16 as the adopted encoding form, implementation level 3. If the adopted ISO/IEC 10646-1 subset is not otherwise specified, it is presumed to be the BMP subset, collection 300. If the adopted encoding form is not otherwise specified, it presumed to be the UTF-16 encoding form."

The third edition adds to the earlier versions in the following areas:

- Regular expressions
- Richer control statements
- Better string and array handling
- Exception handling improvements with `try/catch`
- Internationalization facilities
- Error objects for managing exceptions

ECMAScript edition 3 support is a feature of the MSIE/Jscript 5.5. upgrade and the new Netscape 6.0 release.

See also:	Compliance, Conformance, ECMAScript, Implementation

Cross-references:

ISO/IEC 10646-1 Information Technology – Universal Multiple-Octet Coded Character Set (UCS), plus its amendments and technical corrigenda.

Unicode Technical Report #8, The Unicode Standard, Version 2.1.

ECMAScript version (Standard)

The version history for ECMAScript.

The version history of ECMAScript is slightly behind the current browser implementations. Versions of ECMAScript are defined as editions 2 and 3 of the standard. Edition 1 was superseded by edition 2, which made few functional changes. There were not enough to warrant revising the level of functionality.

ECMA edition 2 is roughly equivalent to JavaScript version 1.1, minus any HTML document modelling.

ECMA edition 3 is roughly equivalent to JavaScript 1.5 and JScript 5.5.

See also:	ECMAScript

Element object (Object/HTML)

A common name for an object that represents an HTML tag or container.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Inherits from:	`Node` object

JavaScript syntax:	IE	*myElement* = document.all(*anIndex*)
	IE	*myElement* = document.all.*anElementID*
	IE	*myElement* = document.all[*anIndex*]
	IE	*myElement* = document.children(*anIndex*)
	IE	*myElement* = document.children.*anElementID*
	IE	*myElement* = document.children[*anIndex*]
	-	*myElement* = *myChildNodes*[*aName*]
	-	*myElement* = *myChildNodes*[*anIndex*]
	-	*myElement* = *myDocument*.getElementById(*anElementID*)
	-	*myElement* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myElement* = *myDocument*.createElement(*aTagName*)
	-	*myElement* = *myDocument*.documentElement
	-	*myElement* = *myDocument*.getElementsByTagName(*aTagName*)[*anIndex*]
HTML syntax:	<*anHTMLTag*>	
Argument list:	*anElementID*	The ID value for the required element
	anHTMLTag	A tag that represents a realizable concrete object
	anIndex	An index location in the collection
	aTagName	The name of an HTML tag to create an element for
	aName	An associative array reference
Object properties:	canHaveChildren, canHaveHTML, className, clientHeight, clientLeft, clientTop, clientWidth, contentEditable, currentStyle, dir, document, firstChild, hideFocus, id, innerHTML, innerText, isContentEditable, isDisabled, isTextEdit, lang, language, lastChild, nextSibling, nodeName, nodeType, nodeValue, offsetHeight, offsetLeft, offsetParent, offsetTop, offsetWidth, outerHTML, outerText, ownerDocument, parentElement, parentNode, parentTextEdit, previousSibling, readyState, recordNumber, runtimeStyle, scopeName, scrollHeight, scrollLeft, scrollTop, scrollWidth, sourceIndex, style, tagName, tagUrn, title, uniqueID	
Object methods:	addBehavior(), applyElement(), blur(), clearAttributes(), click(), componentFromPoint(), contains(), doScroll(), focus(), getAdjacentText(), getAttribute(), getAttributeNode(), getElementsByTagName(), getExpression(), insertAdjacentHTML(), insertAdjacentText(), mergeAttributes(), normalize(), releaseCapture(), removeAttribute(), removeAttributeNode(), removeBehavior(), removeExpression(), replaceAdjacentText(), scrollIntoView(), setAttribute(), setAttributeNode(), setCapture(), setExpression()	
Functions:	handleEvent()	

Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`
Collections:	`all[]`, `attributes[]`, `behaviorUrns[]`, `childNodes[]`, `children[]`, `filters[]`

The standard DOM hierarchy model specifies a variety of properties that should be available across the entire collection of objects. This suggests that all objects in the document would generally behave as if they were sub-classes of a master DOM `element` object.

DOM level 1 describes an `Element` object and an `HTML` object and makes a distinction between the two. JavaScript considers them to be one and the same within a web browser context.

In JavaScript 1.2 and MSIE 4, many HTML tags are represented as objects that have similar methods and properties. It is convenient also to model these as if they were sub-classes of a single super-class. Actually, Netscape at revision 4 does not really support this model very consistently, although it is a sound concept when studying the internals of the MSIE browser. Netscape 6.0 implements a much more robust and complete set of the DOM level 1 capabilities.

The `Element` object described here includes the behavior of the DOM `Element` object with some additional properties and methods that are common to many objects in the various browsers.

We call these `Element` objects. In other reference works they may be called DOM objects, DOM elements or HTML Elements. Trying to work out what kind of element you have is not always easy. Some objects will provide this with their `toString()` method. Others will need some inspection. You can look at `constructor` properties for help. Sometimes `object.constructor.name` will tell you what you need. This is not consistent across all the different kinds of objects, nor is the same technique applicable across all browsers.

By considering all the DOM and HTML Element objects as sub-classes of the `Element` object class, we can document their common behavior collectively and then deal with any specific behavior they may implement on a case by case basis.

We attempt to de-clutter the object descriptions by abstracting properties, methods, collections and events into a super-class wherever possible. This `Element` class does not really exist as a concrete class but it helps to understand the internals of the MSIE and Netscape 6.0 browsers in particular to model it like this. A couple of notable exceptions are the `document` object and `window` object classes which share some properties etc. with the `Element` class, but cannot easily be considered to be sub-classes of the `Element` class. Those properties and methods etc. are used in a contextually different way and may exhibit slightly different behavior, and so they merit being covered in separate (but similar) topics to the ones described as members of the `Element` topic set.

There is another special class and that is the `FormElement` object. These are sub-classes of the fundamental `Element` class, but are a super-class of all `Form` elements which represent `<INPUT>` tags. Many of these share common methods and properties in addition to the `Element` object functionality. Properties, methods, collections and events that relate to the `Input` object class are discussed under topics beginning with the phrase "Input" and are not duplicated here. However, they are listed here in the summary.

If you are inspecting objects with an enumeration loop, you may want to eliminate `Element` object properties so that your enumerator only lists special properties of the object you are inspecting. There is actually no object of the `Element` object type. We explored a great deal of the internal browser model by writing small fragments of JavaScript to walk through the properties of each object type. This yielded some properties that were hitherto undocumented and highlighted some differences between platforms and browsers.

In MSIE, there is a class for each type of HTML tag. There is also an `array` class for collections of each type of tag. For example, for an imaginary tag called `<XXXX>`, there will be an object class with the name `XXXX` which represents tags of that type regardless of whether they are upper or lower cased. There will also be an `XxxxArray` collection for that class which is yielded by this expression:

```
document.all.tags("XXXX")
```

If you create imaginary tags in your documents, MSIE won't render them nor will it create objects to represent them. However, the expression shown above will yield an empty collection of an appropriate type.

Versions of Netscape prior to 6.0 do not implement this mechanism because they manage its content with a different DOM construction technique. Netscape 6.0 and MSIE support a more consistent interface to all of the tags in a document. No version of Netscape provides the `all[]` collection.

If you want to examine the properties of `Element` objects you need to be able to isolate those properties that are inherited and those that are not. The example code for MSIE includes a function that you can use to exclude items that are inherited from `Element` objects as you enumerate the properties of an object. We used this to examine the undocumented structures within web content viewed in the browsers. There is also an attribute accessor function demonstrated here. The rest of the example demonstrates a framework for examining object properties.

The DOM level 2 specification adds the following new methods to support namespaces:

- `getAttributeNS()`
- `setAttributeNS()`
- `removeAttributeNS()`
- `getAttributeNodeNS()`
- `setAttributeNodeNS()`
- `getElementsByTagNameNS()`
- `hasAttribute()`
- `hasAttributeNS()`

DOM level 2 moves the `normalize()` method to the `Node` object but it is still available here through inheritance.

As the DOM standards advance and proliferate and the browser become more DOM standards compliant, a standards based approach to the documentation may become more appropriate. We have structured our coverage around a browser centric approach.

`Element` objects will have other properties and methods not included here because they are not yet implemented or fall outside the standard. The following properties are present in some implementations for example:

- `onOffBehavior`
- `hasMedia`
- `syncMaster`

These are part of the SMIL and HTML+TIME standards which are still in a state of evolution and therefore the behavior of the implementation may change.

Warnings:

- Be careful how you operate on these objects. Traversing the properties in a `for( ... in ... )` loop can recursively lock up your browser for some objects that represent high level parts of the DOM hierarchy.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY >
<IMG ID="IMAGE1" SRC="wrox.gif">
<TABLE CELLPADDING=2 CELLSPACING=2 BORDER=1>
<SCRIPT>
var myObject = document.getElementById("IMAGE1");
document.write(displayProperties(myObject));

// Display a table of enumerated properties
function displayProperties(anObject)
{
   var myOut = "";
   myOut += '<TR BGCOLOR="ANTIQUEWHITE">';
   myOut += "<TD>Object type</TD><TD>";
   myOut += typeof anObject;
   myOut += "</TD><TD>";
   myOut += anObject;
   myOut += "</TD><TD>Name</TD><TD>Specified</TD>";
   myOut += "<TD>Value</TD></TR>";

   for(myProp in anObject)
   {
      // Place a not symbol on this condition to see
      // non inherited properties
      if(isElementProperty(myProp))
      {
         myOut += "<TR><TD>";
         myOut += myProp;
         myOut += "</TD><TD>";
         myOut += typeof anObject[myProp];
         myOut += "</TD><TD>";
         myOut += displayableValue(myProp, anObject[myProp]);
         myOut += "</TD><TD>";
         myOut += getPropAttr(anObject, myProp, "name");
         myOut += "</TD><TD>";
         myOut += getPropAttr(anObject, myProp, "specified");
         myOut += "</TD><TD>";
         myOut += getPropAttr(anObject, myProp, "value");
         myOut += "</TD></TR>";
      }
   }

   return myOut;
}
```

```
// Prevent recursive displays
function displayableValue(aProperty, aValue)
{
   switch(aProperty)
   {
      case "innerHTML"       :
      case "innerText"       :
      case "outerHTML"       :
      case "outerText"       :
         return "**" + aProperty + "**";
   }
   return aValue;
}

// Element object property flag
function isElementProperty(aProperty)
{
   switch(aProperty)
   {
      case "all"             :
      case "attributes"      :
      case "childNodes"      :
      case "children"        :
      case "className"       :
      case "currentStyle"    :
      case "dir"             :
      case "document"        :
      case "filters"         :
      case "firstChild"      :
      case "id"              :
      case "innerHTML"       :
      case "innerText"       :
      case "isTextEdit"      :
      case "lang"            :
      case "language"        :
      case "lastChild"       :
      case "nextSibling"     :
      case "nodeName"        :
      case "nodeType"        :
      case "nodeValue"       :
      case "offsetHeight"    :
      case "offsetLeft"      :
      case "offsetParent"    :
      case "offsetTop"       :
      case "offsetWidth"     :
      case "outerHTML"       :
      case "outerText"       :
      case "ownerDocument"   :
      case "parentNode"      :
      case "parentElement"   :
      case "parentTextEdit"  :
      case "previousSibling" :
      case "sourceIndex"     :
      case "style"           :
      case "tagName"         :
      case "title"           :
```

```
            return true;
      }
      return false;
   }

   // Property attribute accessor
   function getPropAttr(anObject, aProp, anAttrib)
   {
      if(anObject.attributes[aProp])
      {
         return anObject.attributes[aProp][anAttrib];
      }
      else
      {
         return " ";
      }
   }

   </SCRIPT>
   </TABLE>
   </BODY>
   </HTML>
```

See also:	`Attributes` object, `Document.all[]`, `Document.createElement()`, `Document.documentElement`, `isElementProperty()`, `Node` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`canHaveChildren`	-	5.0 +	-	5.0 +	-	-	-	-
`canHaveHTML`	-	5.5 +	-	5.5 +	-	-	-	ReadOnly
`className`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`clientHeight`	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
`clientLeft`	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
`clientTop`	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
`clientWidth`	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
`contentEditable`	-	5.5 +	-	5.5 +	-	-	-	-
`currentStyle`	-	5.0 +	-	5.0 +	-	-	-	Warning
`dir`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`document`	-	3.0 +	-	4.0 +	5.0 +	-	-	Warning, ReadOnly
`firstChild`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`hideFocus`	-	5.5 +	-	5.5 +	-	-	-	-
`id`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`innerHTML`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
`innerText`	-	3.0 +	-	4.0 +	-	-	-	Warning

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
isContentEditable	-	5.5 +	-	5.5 +	-	-	-	ReadOnly
isDisabled	-	5.5 +	-	5.5 +	-	-	-	ReadOnly
isTextEdit	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
lang	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
language	-	3.0 +	-	4.0 +	-	-	-	-
lastChild	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
nextSibling	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
nodeName	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	ReadOnly
nodeType	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	ReadOnly
nodeValue	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
offsetHeight	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
offsetLeft	-	3.0 +	-	4.0 +	5.0 +	-	-	Warning, ReadOnly
offsetParent	-	3.0 +	-	4.0 +	5.0 +	-	-	Warning, ReadOnly
offsetTop	-	3.0 +	-	4.0 +	5.0 +	-	-	Warning, ReadOnly
offsetWidth	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
outerHTML	-	3.0 +	-	4.0 +	-	-	-	Warning
outerText	-	3.0 +	-	4.0 +	-	-	-	Warning
ownerDocument	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	ReadOnly
parentElement	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
parentNode	1.5 +	5.0 +	6.0 +	5.0 +	5.0 +	1 +	-	ReadOnly
parentTextEdit	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
previousSibling	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
readyState	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
recordNumber	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
runtimeStyle	-	5.0 +	-	5.0 +	-	-	-	-
scopeName	-	5.0 +	-	5.0 +	-	-	-	-
scrollHeight	-	5.0 +	-	5.0 +	-	-	-	Warning, ReadOnly
scrollLeft	-	5.0 +	-	5.0 +	-	-	-	Warning
scrollTop	-	5.0 +	-	5.0 +	-	-	-	Warning
scrollWidth	-	5.0 +	-	5.0 +	-	-	-	Warning, ReadOnly
sourceIndex	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
style	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	-	-
tagName	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	1 +	-	ReadOnly
tagUrn	-	5.0 +	-	5.0 +	-	-	-	-
title	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
uniqueID	-	5.0 +	-	5.0 +	-	-	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
addBehavior()	-	5.0 +	-	5.0 +	-	-	-	-
applyElement()	-	5.0 +	-	5.0 +	-	-	-	Warning
blur()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
clearAttributes()	-	5.0 +	-	5.0 +	-	-	-	Warning
click()	-	3.0 +	-	4.0 +	-	-	-	-
componentFromPoint()	-	5.0 +	-	5.0 +	-	-	-	-
contains()	-	3.0 +	-	4.0 +	5.0 +	-	-	-
doScroll()	-	5.0 +	-	5.0 +	-	-	-	-
focus()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
getAdjacentText()	-	5.0 +	-	5.0 +	-	-	-	-
getAttribute()	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	1 +	-	Warning
getAttributeNode()	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
getElementsByTagName()	1.5 +	5.0 +	6.0 +	5.0 +	5.0 +	1 +	-	Warning
getExpression()	-	5.0 +	-	5.0 +	-	-	-	-
insertAdjacentHTML()	-	3.0 +	-	4.0 +	-	-	-	Warning
insertAdjacentText()	-	3.0 +	-	4.0 +	-	-	-	Warning
mergeAttributes()	-	5.0 +	-	5.0 +	-	-	-	-
normalize()	1.5 +	-	6.0 +	-	-	1 +	-	-
releaseCapture()	-	5.0 +	-	5.0 +	-	-	-	-
removeAttribute()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
removeAttributeNode()	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
removeBehavior()	-	5.0 +	-	5.0 +	-	-	-	-
removeExpression()	-	5.0 +	-	5.0 +	-	-	-	-
replaceAdjacentText()	-	5.0 +	-	5.0 +	-	-	-	-
scrollIntoView()	-	3.0 +	-	4.0 +	-	-	-	-
setAttribute()	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	1 +	-	-
setAttributeNode()	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
setCapture()	-	5.0 +	-	5.0 +	-	-	-	-
setExpression()	-	5.0 +	-	5.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Node` object

Element.addBehavior() (Method)

Attach a behavior to an object in MSIE.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myElement*`.addBehavior()`

MSIE supports an extension to style sheet handling that provides for scripts to be called as part of the style. These are called behaviors. The script source text is stored in a special and separate file with an `.htc` (or `.HTC`) file extension. It stands for HTml Component.

This augments the already available event handler support which provides for handlers to be registered with individual objects or with a particular instance of a tag. Behaviors allow a handler to be registered with all occurrences of a tag with only one registration being necessary.

Here is the contents of a simple HTC file taken from the Wrox book *Professional JavaScript*.

```
<SCRIPT LANGUAGE="JScript">attachEvent("onclick", event_onclick);function
event_onclick(){   alert("Read the manual.");}</SCRIPT>
```

This can be defined in a style sheet or in a `<STYLE>` tag. Here is how it would be invoked with an inline `<STYLE>` defined in the `<HEAD>` of a document:

```
<STYLE>   .help{behavior:url(help.htc)}</STYLE>
```

Having defined the style, you can now apply this to any element. For example, it could apply to the entire contents of a `<DIV>` block like this:

```
<DIV CLASS="help" DELAY="2000">Click for help</DIV>
```

The `DELAY` HTML tag attribute provides a way to delay the presentation of the `<DIV>` block contents until the browser has had a chance to load the HTC file content.

There are more complex ways to use behaviors that involve the use of XML mark-up and which can avoid the use of `attachEvent()` methods. Refer to the Microsoft web site for further details.

See also:	`.htc`, Behavior, `Element.removeBehavior()`

Web-references:

http://msdn.microsoft.com/workshop/essentials/versions/IE5behave.asp

Element.all[] (Collection)

An array containing references to all elements within this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Collection` object	
JavaScript syntax:	IE	*`myElement`*`.all`

This property yields a collection of objects that are descendants within the DOM hierarchy of the receiver.

The collection (Array) contains a reference to all the objects that represent the contents of the HTML contained within the current `Element` object. If the `Element` object is the document, then this array will contain every single object within the document. If it represents a `<TABLE>` then it will contain references to objects within that table and none that live outside it.

The objects are listed strictly in the order in which they appear in the document source.

You can access the items by index number or you can operate on the collection with the `item()` or `tags()` methods. If you use index number access, MSIE allows you to use either square brackets or parentheses to delimit the index value. Associative references may also work if the target item has a `NAME` or `ID` value defined.

Refer to the topic that describes the `Collection` object for further details.

The event handling is also dealt with in a generic manner via the `Element` object. However some object sub-classes also support additional event types. Event types and handlers are dealt with in topics pertaining to each type of event.

Warnings:

- Be careful that when you traverse this collection you avoid the possibility of creating a recursive reference. This can lead to a script lockup that could crash your browser.
- Netscape does not support the `all[]` collection for document objects. Versions prior to Netscape 6.0 also do not consistently support access via an `ID` value.

See also:	`Anchor` object, `Collection` object, `Document.all[]`, `Document.anchors[]`, `Document.forms[]`, `Document.images[]`, `Document.links[]`, `Element` object, `Form` object, `FormElement` object, Hierarchy of objects, `HyperLink` object, `Image` object, `LINK` object

Property attributes:

`ReadOnly`.

Element.applyElement() (Method)

A way of defining an `Element` object as the child or parent of another element.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myElement*.applyElement(*anObject*)
	IE	*myElement*.applyElement(*anObject*, *aLocation*)
Argument list:	*anObject*	An object to be referenced
	aLocation	The relationship to construct

When the element is applied, you must specify the `element` object to which it is to become attached.

There is an optional second parameter to define whether it should become the parent or the child of that object. If the second parameter is not specified, then the element becomes the parent of the target object.

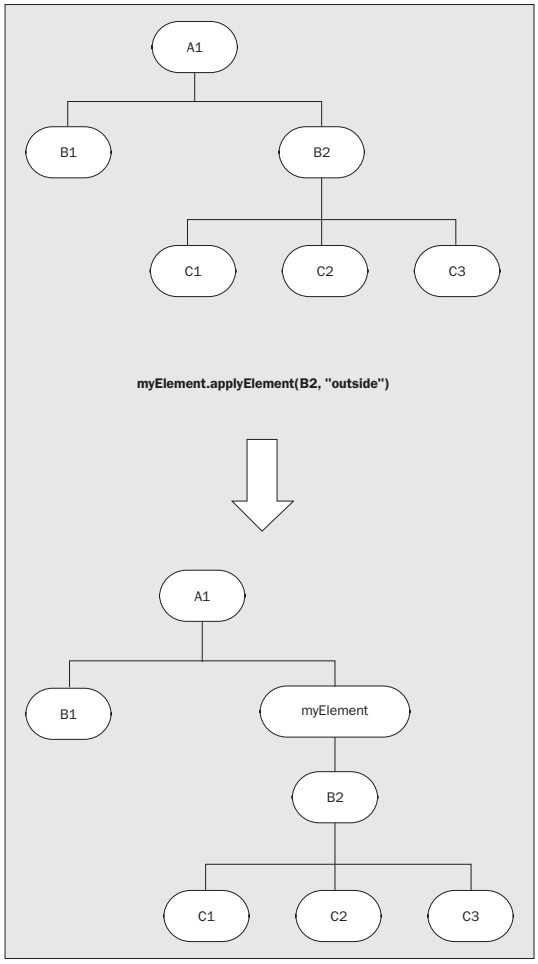
The second parameter can be any of these values:

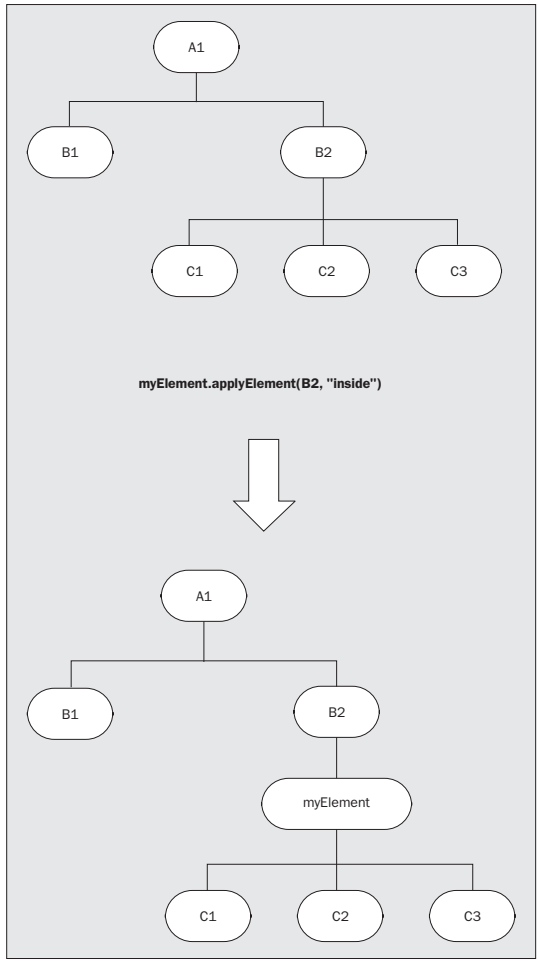
- `outside`
- `inside`

Warnings:

- Although this can be used at run-time when the page is being constructed, any removal of elements during this process may leave the page in a state where it cannot be properly rendered.

See also:	`Function.apply()`





Element.attributes[] (Collection)

A reference to a collection of `attribute` objects for the HTML tag that the `Element` object represents.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Attributes object`	
JavaScript syntax:	-	*`myElement`*`.attributes`

This property yields an `attributes` object which is a collection of `attribute` objects belonging to the object instantiated by an HTML tag.

The `attributes` object contains properties that JavaScript can inspect to obtain the value of HTML tag attributes for the tag that instantiated the `Element` object.

Structurally, the `attributes` object is an array whose elements correspond to properties of the `Element` object, each containing a name, a value and a flag. The `Flag` value is set `true` only when the property has a meaningful value. That value is reflected in the `value` property of the `attribute` object.

So, to find out all about an `Element` object property, you can get the value of the property by enquiring for its contents directly. However, if you use the property name to index the `attributes` array for that `Element` object, you can find out other information about that property by inspecting the `attribute` properties.

Access a property of an object like this:

myObject`["propertyName"]`

or like this:

myObject`.propertyName`

To access its corresponding attributes use this:

myObject`.attributes["propertyName"].name`

myObject`.attributes["propertyName"].specified`

myObject`.attributes["propertyName"].value`

However you should test for the existence of the *myObject*`.attributes[`*myProp*`]` object first because an `attribute` object is not created for every property of an `Element` object but only those that correspond to valid HTML tag attributes for the parent tag.

Any `Element` objects that represent HTML tags will have a `nodeType` of 1. Other node types supported by their `attributes` property will contain the null value.

The example code shows how this attribute access might be provided with a function.

Example code:

```
// An attribute accessor function
function getPropertyAttribute(anObject, aProp, anAttrib)
{
   if(anObject.attributes[aProp])
   {
      return anObject.attributes[aProp][anAttrib];
   }
   else
   {
      return "";
   }
}
```

See also:	Attributes object, Element object, Element.mergeAttributes(), Node.attributes[]

Property attributes:

ReadOnly.

Element.behaviorUrns[] (Collection)

A collection of all the behaviors attached to an element (as a set of URN strings).

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myElement*.behaviorUrns

This lists all the URN values that point at HTC files from within the current document.

Property attributes:

ReadOnly.

Element.canHaveChildren (Property)

A Boolean flag indicating whether this element may have child Nodes associated with it.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	IE	*myElement*.canHaveChildren

If an element is able to own child Nodes and hence populate its childNodes[] collection and its children[] and all[] collections, then this property should be set true.

Element.canHaveHTML (Property)

A flag property that indicates whether the element can contain HTML.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	Boolean primitive	
JavaScript syntax:	IE	*myElement*.canHaveHTML

If an element is able to respond to `innerHTML` set and get accessors and hence display some visible onscreen content, this property should yield `true`.

See also:	`Element.innerHTML`, `Element.innerText`

Property attributes:

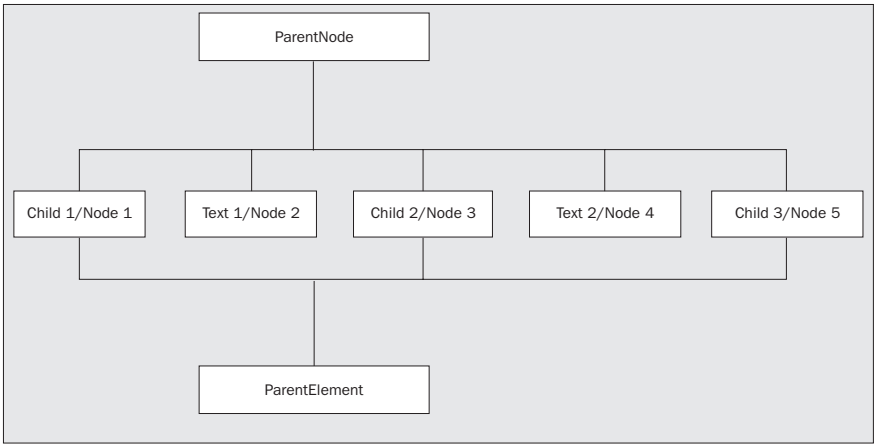
`ReadOnly`.

Element.childNodes[] (Collection)

A list of object nodes within the DOM that are direct children of the `node` object that owns this collection.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`ChildNodes` object
JavaScript syntax:	- *myElement*`.childNodes`

This is part of the internal management of the DOM that browsers use to manage the document content. There are additional nodes here that map to fragments of text in between HTML tags. So there may be more items in this array than in the `children[]` property.



See also:	`ChildNodes` object, `Element` object, Hierarchy of objects, `Node.childNodes[]`

Property attributes:

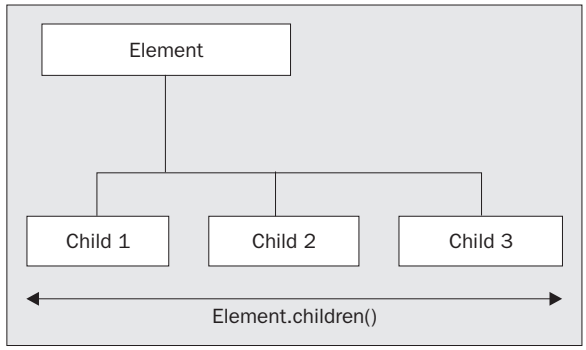
`ReadOnly`.

Element.children[] (Collection)

The elements that are direct children of this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Collection` object	
JavaScript syntax:	IE	*myElement*`.children`

A collection of objects which are the immediate children of the receiving `Element` object. These objects generally represent HTML tags but there are other objects that are linked in to the DOM hierarchy that represent the fragments of text between the HTML tags.



See also:	`Collection` object, `Element` object, Hierarchy of objects

Property attributes:

`ReadOnly`.

Element.className (Property)

The value of the `CLASS` tag attribute for the HTML tag that represents this element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myElement*`.className`
	-	*myElement*`.className = `*aClassName*
Argument list:	*aClassName*	A new class name that exists in the cascading style sheet

This property contains the name of the cascading style sheet class for the tag that the `Element` object represents. Because MSIE implements its tag objects so consistently as sub-classes of the `Element` object class, it is possible to apply `CLASS` tag attributes to tags that really shouldn't have style sheets associated with them. The `<HTML>` and `<HEAD>` tags for example. Nevertheless, you can still define a `CLASS` tag attribute and its value will be reflected and accessible in this property.

You can read or write this value to manipulate the style attributes of the object.

The value is specified as a String primitive, but is case sensitive. If you specify multiple items, they must be space separated.

Changing this value can be a way to change a whole range of style settings on an object in a single hit rather than modifying the individual settings one by one.

Warnings:

- This property used to be called `Element.class` but has been renamed as of DOM level 1 to avoid namespace conflicts with a possible future `object.class` property.
- This is not the name of the object class that is associated with the tag. In Netscape, you can usually figure out the class of an object by accessing the `constructor` property to get an object that represents the class and then you can obtain the `name` property of the constructor to find the class name. This doesn't work in MSIE in general. However, you can get a string version of an object in MSIE in many cases that is of the form "[object XXXXX]" where XXXXX is the class name. You will need to dismantle the string with a fragment of script.
- There can be a conflict of style settings where the same attribute is controlled from several class settings. The following precedence rules are applied:

  1. Styles defined in the Element's HTML tag.
  2. Styles applied to the Element with the `CLASS` HTML tag attribute.
  3. Styles applied using the `ID` value of an Element.
  4. Inline styles defined within the script environment.

See also:	`CLASS="...",Element` object

Element.clearAttributes() (Method)

Clear the attributes from an `element` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myElement*`.clearAttributes()`

Any attribute values that are associated with an element can be cleared using this method.

Warnings:

- This may not be supported on all platforms that MSIE runs on. In particular, under testing it failed to work on a Macintosh and generated a run-time error.

Element.click() (Method)

Sends a click event to the receiving object to trigger its `onclick` event handler.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	IE	*`myElement`*`.click()`

This is supported by most objects on MSIE but only `input` objects on Netscape, and this is still true in version 6.0.

It simulates the effect of a mouse click on the object. This should trigger the event handler for the `onclick` event. That handler would be a function object and a reference to it should be available in the `onclick` property of the receiving object.

The handler function may have been registered by assigning the function object reference to the property user script control or it may have been registered with the `onClick` HTML tag attribute in the tag that instantiates the object.

See also:	`onClick`

Element.clientHeight (Property)

The height in pixels of the object on the client display surface.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.clientHeight`

The height of the extent rectangle around the `element` object as it appears on screen. This value is measured in pixels.

Warnings:

- This property does not operate consistently across different platforms, and in any case is only available in MSIE. This somewhat limits its usefulness.

Property attributes:

`ReadOnly`.

Element.clientLeft (Property)

The offset in pixels of the object on the client display surface relative to its parent object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.clientLeft`

The pixel coordinate of the left edge of the extent rectangle around the `element` object as it appears on screen.

Warnings:

- This property does not operate consistently across different platforms and in any case is only available in MSIE. This somewhat limits its usefulness.

Property attributes:

`ReadOnly`.

Element.clientTop (Property)

The offset in pixels of the object on the client display surface relative to its parent.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.clientTop`

The pixel coordinate of the top edge of the extent rectangle around the `element` object as it appears on screen.

Warnings:

- This property does not operate consistently across different platforms and in any case is only available in MSIE. This somewhat limits its usefulness.

Property attributes:

`ReadOnly`.

Element.clientWidth (Property)

The width in pixels of the object on the client display surface.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.clientWidth`

The width of the extent rectangle around the `element` object as it appears on screen. This value is measured in pixels.

Warnings:

- This property does not operate consistently across different platforms and in any case is only available in MSIE. This somewhat limits its usefulness.

Property attributes:

`ReadOnly`.

Element.componentFromPoint() (Method)

Determines what object is under a particular pixel coordinate in the window.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Element` object	
JavaScript syntax:	IE	*`myElement`*`.componentFromPoint(`*`anX, aY`*`)`
Argument list:	*`anX`*	A horizontal coordinate
	`aY`	A vertical coordinate

Return a reference to an `element` object that is the upper most in the Z order at the corresponding X-Y coordinate point measured in pixels relative to the upper left of the receiving object.

Element.contains() (Method)

A test to see if the element contains another element.

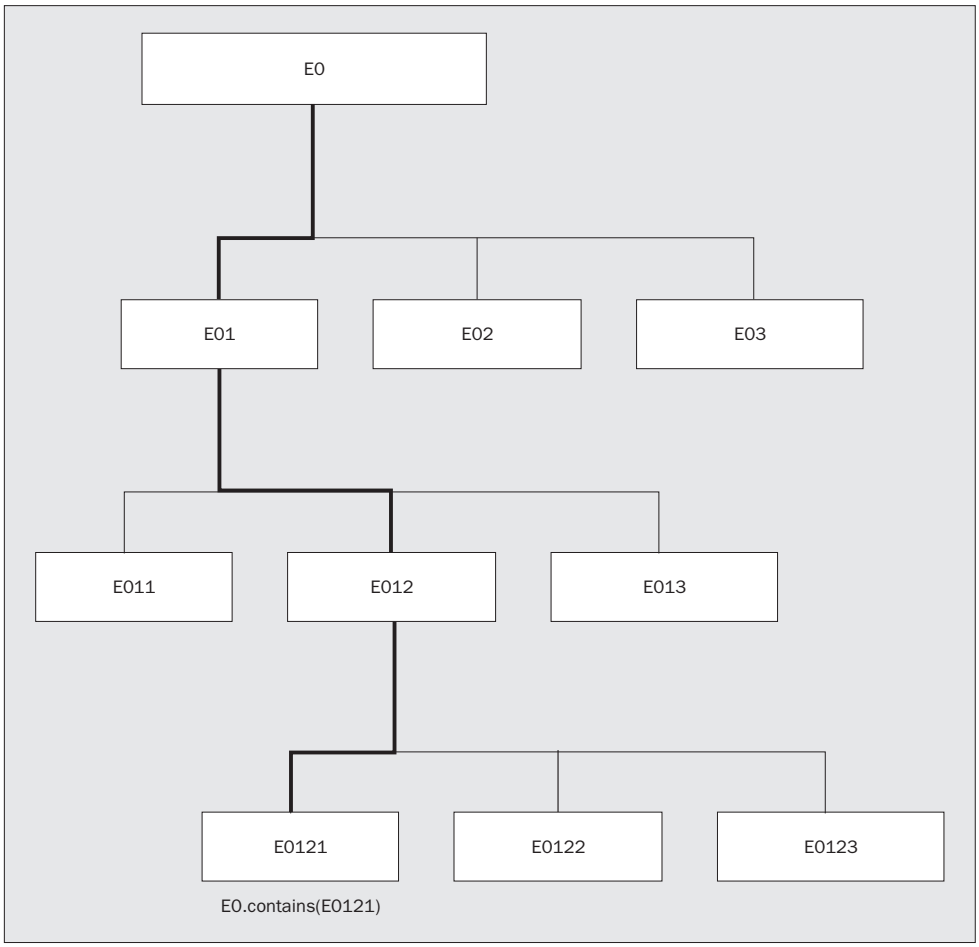
Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0
Property/method value type:	`Boolean primitive`

JavaScript syntax:	IE	*myElement*.contains(*anElement*)
Argument list:	*anElement*	An Element object to check for

The receiving element hierarchy is checked to see if the element that is passed as an argument exists by reference within it.

The result of this method will be `true` if the element referred to by its argument is contained in the receivers node descendants. If by traversing the child nodes of the receiver the indicated element cannot be found, a `false` value is returned.

The functionality of this method is similar to that provided by the DOM level 1 compliant `Node.hasChildNodes()` method which should provide a more portable solution.



See also:	Element object, Node.hasChildNodes()

Element.contentEditable (Property)

A property that determines whether the content of an element can be changed.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myElement*`.contentEditable`
	IE	*myElement*`.contentEditable = `*aSetting*
Argument list:	*aSetting*	One of `true`, `false` or `inherit`

This property can be set under script control. The following values are appropriate for use with it:

- `inherit`
- `true`
- `false`

When set to `false`, the content cannot be changed. When set to `inherit`, the ability to change the content depends on the setting of this property in the parent Node.

You cannot set this property on the following element types:

- TABLE
- COL
- COLGROUP
- TBODY
- TD
- TFOOT
- TH
- THEAD
- TR

However you can work around this by placing `<DIV>` and `<SPAN>` elements inside the table cells and then setting their `contentEditable` property as needed.

Element.currentStyle (Property)

A reference to a `style` object that contains the current style settings for the `Element` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0
Property/method value type:	`Style` object

JavaScript syntax:	IE	*myElement*.currentStyle
	IE	*myElement*.currentStyle = *aStyleObject*
Argument list:	*aStyleObject*	A new style object to associate with the element

Because the style values are cascaded from style sheet to style sheet and may include some inline styles as well as some explicit styles, objects need to maintain a current style value that is the result of all the inheritances applied on top of one another.

In addition they maintain a run-time style which reflects dynamic changes as well. The run-time style is based on the current style in the first place.

The value of this property is a reference to a `style` object. However it does not refer to the same `style` object as the `style` property. You can test for this with the `isObjectEqual()` function that we have covered elsewhere.

Warnings:

- Note that in MSIE, the object that is returned is of the class "`style`" rather than "`Style`". The upper and lower case spelling may differ between browsers. However since you are likely to access the object by reference you may never need to know the specific class name.

See also:	`currentStyle` object, `Element` object, `Element.runtimeStyle`, `Element.style`, `isObjectEqual()`, `runtimeStyle` object, `style` object (2)

Element.dir (Property)

The text direction of the `Element` object content.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myElement*.dir
	-	*myElement*.dir = *aDirection*
Argument list:	*aDirection*	A value that indicates leftwards or rightwards text drawing

The `dir` property may be set to indicate a left to right or right to left parsing direction.

This is part of the localization support and represents the contents of the `DIR` tag attribute.

If you assign a value to this property it is case sensitive and must be either "`ltr`" or "`rtl`".

This property works in conjunction with the `lang` property to control the direction of text flow.

On MSIE 5.0 for Macintosh, this can reverse the order of the table columns so they are drawn right to left, but present the items in the cells still written left to right. This may be because tables block the inheritance of style attributes from the outer containing elements into the table cells.

See also:	`BDO.dir`, `Element` object, `NOFRAMES.dir`, `NOSCRIPT.dir`

Element.document (Property)

The document containing this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Document` object	
JavaScript syntax:	IE	*`myElement`*`.document`

This is the document in which the `Element` object is instantiated. Although this is not generally available as an object property in Netscape, it is supported by the `Window` object by virtue of it being a global value. This means that for most cases, the scope chain adds the `document` object to all objects, or at least it appears as if they have the property. However, in Netscape, the `document` property is shared by all objects in the window rather than each having its own private `document` property that refers to the same `document` object. The difference is subtle.

Warnings:

- Be careful when blindly using the properties of `Element` objects, as a property such as this can lead you into a recursive loop situation.

See also:	`Doctype` object, `Document` object, `Element` object, `Element.ownerDocument`, `Layer.document`, `Window.document`

Property attributes:

`ReadOnly`.

Element.doScroll() (Method)

Simulates a mouse click on a scroll bar.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*`myElement`*`.doScroll()`
	IE	*`myElement`*`.doScroll(`*`aKeyword`*`)`
Argument list:	*`aKeyword`*	Define the scroll item to click on

This emulates the action of the user having clicked on a scrollbar item with the mouse. The scroll controller that is clicked on depends on the keyword used in the method call.

Keyword:	Description:
`down`	Composite reference to `scrollbarDown`.
`left`	Composite reference to `scrollbarLeft`.
`pageDown`	Composite reference to `scrollbarPageDown`.
`pageLeft`	Composite reference to `scrollbarPageLeft`.
`pageRight`	Composite reference to `scrollbarPageRight`.
`pageUp`	Composite reference to `scrollbarPageUp`.
`right`	Composite reference to `scrollbarRight`.
`scrollbarDown`	Down scroll arrow is at the specified location.
`scrollbarHThumb`	Horizontal scroll thumb or box is at the specified location.
`scrollbarLeft`	Left scroll arrow is at the specified location.
`scrollbarPageDown`	Page-down scroll bar shaft is at the specified location.
`scrollbarPageLeft`	Page-left scroll bar shaft is at the specified location.
`scrollbarPageRight`	Page-right scroll bar shaft is at the specified location.
`scrollbarPageUp`	Page-up scroll bar shaft is at the specified location.
`scrollbarRight`	Right scroll arrow is at the specified location.
`scrollbarUp`	Up scroll arrow is at the specified location.
`scrollbarVThumb`	Vertical scroll thumb or box is at the specified location.
`up`	Composite reference to `scrollbarUp`.

The default value is `scrollbarDown`.

Element.filters[] (Collection)

Filters are supported by MSIE and provide some stylistic control over presentation. This collection contains all the filters associated with an `Element`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Filters` object	
JavaScript syntax:	IE	*`myElement`*`.filters`

This is a reference to a filter collection object that contains details of objects that represent a DHTML filter effect that is invoked during event handling.

See also:	Behavior, `Element` object, Filter, `Filter` object, `style.filter`, Transition

Property attributes:

`ReadOnly`.

Element.firstChild (Property)

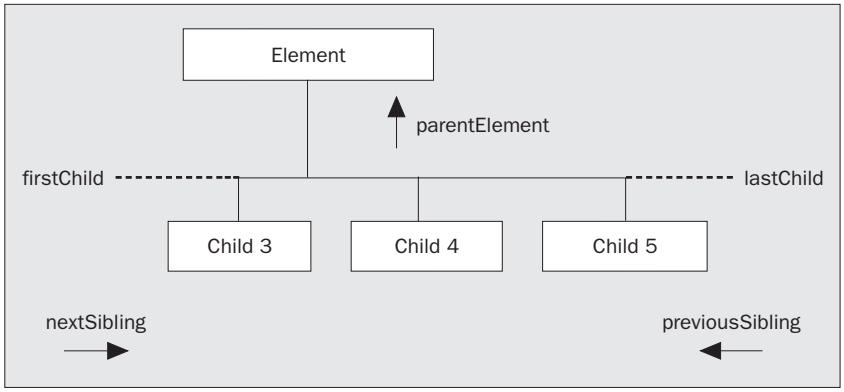
The first object in the collection of direct children of this element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`Element` object
JavaScript syntax:	- *`myElement`*`.children[0]` - *`myElement`*`.firstChild`

The `Element` objects are instantiated as the HTML tags are parsed within the document source. They are added to several collections and can be navigated in a variety of ways.

Each `Element` object has an array of `Element` objects that are considered to be its direct descendants (that is children). This collection will not contain its children's children. The first element in this collection can be referred to by index and its siblings by means of an enumerator or `for( ... in ... )` loop. However you can access it directly with this property.

If the element has no children, then this property will contain a null value.



See also:	`Element` object, `Element.lastChild`, `Element.nextSibling`, `Element.parentElement`, `Element.previousSibling`, `Node.firstChild`, `Node.hasChildNodes()`

Element.getAdjacentText() (Method)

Obtains the text adjacent to the object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myElement*`.getAdjacentText(`*aRelativePosition*`)`
Argument list:	*aRelativePosition*	An indication of where the new HTML is to be placed

Using the same locational techniques as `insertAdjacentText()`, this method retrieves the text adjacent to the element.

See also:	`Element.insertAdjacentHTML()`, `Element.insertAdjacentText()`, `Element.replaceAdjacentText()`

Element.getAttribute() (Method)

An accessor for reading named custom attributes.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myElement*`.getAttribute(`*anAttribName*`)`
	-	*myElement*`.getAttribute(`*anAttribName*`,` *aCaseSense*`)`
Argument list:	*aCaseSense*	A flag to indicate a case-sensitive lookup
	anAttribName	An attribute of an `Element` object

This is an accessor method which is used to access named attributes of an `Element` object. Attributes are not properties in the strict sense of the word, but may be accessible as if they were in some implementations.

This accessor is intended to provide a means of managing custom attributes.

It would be logical to assume that attributes are named uniquely but if several share the same name, differing only in case sensitivity, then if a case-insensitive search is used you may not retrieve the one you expect. It is likely that you'll be given the last one that occurs, but this may be implementation-dependent.

The case-sensitive flag should be set to the Boolean `true` value to force a case-sensitive search and `false` to ignore the case of letters in the attribute name.

The following values can be passed in the optional case-sensitive flag argument:

- `true` – A case-sensitive search is carried out.
- `false` – A case-insensitive search is carried out.
- 0 – A case-insensitive search of property values is carried out by default. If several instances are located, then only the last is returned.
- 1 – A case-sensitive search is carried out.
- 2 – The value is returned exactly as was originally defined in the document source regardless of subsequent `setAttribute()` calls.

The result will be value of the attribute. If the element does not have an attribute of the specified name, a `null` value is returned.

Warnings:

- If a case-sensitive search is carried out using a property name stored in a variable, you should make sure that the same setting was defined for a corresponding `setAttribute()` call. If you don't, then it is possible that the name may have a case change if the 0 value was used in the `setAttribute()` call. After that case change, the value in the variable will no longer match the property defined for the receiving object.

See also:	`Element` object, `Element.removeAttribute()`, `Element.setAttribute()`,`style.getAttribute()`

Element.getAttributeNode() (Method)

Given its name, this will retrieve an `Attribute` node for the `Element` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Attribute` object	
JavaScript syntax:	-	*`myElement`*`.getAttrbuteNode(`*`aName`*`)`
Argument list:	*`aName`*	The name of an attribute node

See also:	`Attribute` object

Element.getElementsByTagName() (Method)

Obtain a collection of elements that have the same tag name.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`NodeList` object	
JavaScript syntax:	-	*myElement*`.getElementsByTagName(`*aTagName*`)`
Argument list:	*aTagName*	The name of a tag

Modern browsers are implementing a DOM standard architecture that represents a document in the same way regardless of the browser you are running. This means that the focus of script development will move away from a browser-centric view to a standards-centric approach. Using DOM will lead to you developing a much more portable script product.

One of the important "gateway" methods to this DOM model is `getElementsByTagName()`. This can be applied to a document or to an individual HTML element within it.

The `getElementsByTagName()` method returns a collection of elements that are children of the receiving element so long as they were instantiated by the specified tag name.

Many HTML tags create a container within which other tags are placed. `<HTML>` contains `<HEAD>` and `<BODY>` as its immediate children. A `<BODY>` tag might contain a `<TABLE>` which contains `<TR>` row elements, each of which has its own collection of `<TD>` elements. Within those table cells, further parent- child relationships are constructed.

The `getElementsByTagName()` method will walk down that parent-child tree structure and collect a reference to each `Element` object that was instantiated by the HTML tag name you specify. Then on return, it passes back a `Collection` object with them organized for easy access with the `item()` method or by array index.

Since you can apply this search to an individual HTML element, the traversal of the document tree can be very efficient because you only need to walk the portion local to the objects you want to find.

Warnings:

- Netscape 6.0 is forgiving of non-existent tag names. It will instantiate an `Element` object and you can access it using this DOM method even though the tag is not a valid HTML item. MSIE does not do that. It won't parse incorrect tags into DOM objects and therefore this is not a portable workaround for hiding data in documents.
- Beware that `item()` methods that are used to access individual members of a collection come in several varieties with different argument values. In the case of the MSIE `Collection` object, the method name is capitalised so you need to call `Item()` and not `item()` on those collections.

See also:	`Document.getElementsByTagName()`, `NodeList` object

Element.getExpression() (Method)

Obtains an expression value from within a style rule.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myElement`*`.getExpression()`

The rules that are used to construct a style are comprised of multiple expressions. You can use this method to extract the value of an expression from a style item.

Given the rule might contain a line such as:

```
width:200px
```

the `getExpression()` method can be applied to the `style` object containing the rule with that expression like this:

```
myStyle.getExpression("width:")
```

The value 200px would be returned.

See also:	`Element.removeExpression(), Element.setExpression(),style.getExpression(), style.removeExpression(),style.setExpression()`

Element.hideFocus (Property)

This property controls whether there is any visible effect of acquiring focus on an object.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myElement`*`.hideFocus`
	IE	*`myElement`*`.hideFocus = `*`aSwitch`
Argument list:	*`aSwitch`*	A Boolean value

This simply controls the visible effect of having focus. To control whether an object can receive focus or not, the `tabIndex` property should be modified instead.

Setting the property `true` inhibits the visible focus effect. Setting it `false` restores the visible effect of receiving focus to the normal behavior.

See also:	`onFocus`

Element.id (Property)

The value of the `ID` tag attribute in the HTML tag that represents this element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *`myElement`*`.id`
HTML syntax:	`<ELEMENT_TAG ID="anIDValue">`
Argument list:	*`anIDValue`* A valid (and unique within the document) ID value

DOM level 1 specifies this property as a member of the `HTMLElement` object.

HTML elements in MSIE are often identified using their `ID` value. This is a textual name added with the `ID` HTML tag attribute. Netscape 6.0 should support this more reliably than earlier versions of the browser.

The value of the `ID` HTML tag attribute is stored in this property. This is also used as a means of locating elements by `ID` value.

You can change the `ID` value of an element by setting this property since it has read and write access. However it is hard to think of a circumstance where that might be useful. It is really meant to be a read-only property.

DOM provides some help when searching for elements and you can use the `getElementsById()` method on the `document` object in a DOM compliant implementation. Note that a `getElementsById()` method is not supported on the individual HTML elements. This does mean that DOM compliant searches by `ID` will traverse the entire document if necessary.

DOM provides several other mappings and you can traverse a node hierarchy as well, so this is not likely to prove a severe limitation.

Warnings:

- You must make sure that any value you assign to this property is unique within the document since it is used as part of the document navigation mechanism.
- If you use the same `ID` value for more than one element, it is not clear which one will be located first when the document hierarchy is traversed, although it is likely to be the first one that was defined.
- In the case of objects with the same `ID`, you may be presented with a collection or array of objects rather than a single object. This can make it difficult to write general purpose scripts.

See also:	`Element` object, `ID="..."`

Element.innerHTML (Property)

The inner HTML of a tag in the document.

Availability:	JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myElement*.innerHTML
	-	*myElement*.innerHTML = *someHTML*
Argument list:	*someHTML*	Some new HTML content to place inside the object

The HTML in between but not including the tags that define the element is extracted. The resulting content complete with tags and text is returned when this property is being read.

When the property is being written to, the start and end tags remain intact but everything between them is replaced with a sequence of HTML content.

This property contains the text and any HTML tags that are contained in the body of the element. If you only want the text, then use the `innerText` property.

If you want to include the containing tags, then use the `outerHTML` or `outerText` properties instead.

You can assign new text to be displayed in the HTML element by using this property as an LValue.

The `innerHTML` property of an HTML `Element` object is supported in Netscape 6.0 although it is not mandated in the DOM specification. However Netscape 6.0 does not also support `innerText`, `outerHTML` or `outerText`. You should be able to accomplish what you need with `innerHTML` though.

Warnings:

- Be careful if you extract the `innerHTML` of an element and `document.write()` it back to the same document. You can create recursive loop situations if you are evaluating in global code during the document loading process.
- You cannot set this property while the document is loading.
- MSIE version 4 for Macintosh does not support this as widely as the Windows version or later versions of MSIE for Macintosh.

See also:	`Element` object, `Element.canHaveHTML`, `Element.innerText`, `Element.insertAdjacentHTML()`, `Element.outerHTML`, `Element.outerText`

Element.innerText (Property)

The inner text of a tag in the document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myElement*`.innerText`
	IE	*myElement*`.innerText = `*aString*
Argument list:	*aString*	Text to replace the inner text

The HTML in between but not including the tags that define the element is extracted and any tags are removed. The resulting text is returned when this property is being read.

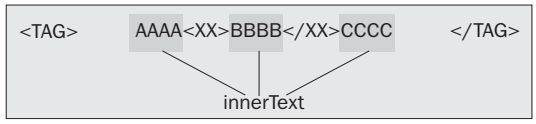
When the property is being written to, the start and end tags remain intact but everything between them is replaced with a text string. To place HTML tags between the tags, use the innerHTML property.

For some objects, this is equivalent to the text property that Netscape supports.

It is effectively the same as `anElement.innerHTML` as long as there is no mark-up on the text between the beginning and ending tags.

If you want to include the containing tags, then use the `outerText` or `outerHTML` properties instead.

You can assign new text to be displayed in the HTML element by using this property as an LValue.



Warnings:

- This is not supported by Netscape although the `text` property may be supported by some objects. You can probably do all you need to do by means of the `innerHTML` property which Netscape 6.0 does support.
- You will need to detect the browser type before attempting to use this property.
- Be careful if you extract the `innerText` of an element and `document.write()` it back to the same document. You can create recursive loop situations if you are evaluating in global code during the document loading process.
- You cannot set this property while the document is loading.
- MSIE version 4 for Macintosh does not support this as widely as the Windows version or later versions of MSIE for Macintosh.
- Netscape does not support it at all.

See also:	Anchor.text, Element object, Element.canHaveHTML, Element.innerHTML, Element.insertAdjacentHTML(), Element.outerHTML, Element.outerText

Element.insertAdjacentHTML() (Method)

Inserts some HTML into the document adjacent to this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	undefined	
JavaScript syntax:	IE	*myElement*.insertAdjacentHTML (*aRelativePosition*, *someHTML*)
Argument list:	*aRelativePosition*	An indication of where the new HTML is to be placed
	someHTML	The new fragment of HTML to be inserted

This is quite similar to an expression that assigns a new value to the `innerHTML` property of an `Element` object. In this case, the new HTML leaves the existing HTML intact.

The relative positions where the adjacent HTML can be introduced determine where the new HTML is to be inserted. The following relative positions can be specified:

Position:	Description:
AfterBegin	Immediately after the opening tag and before any other content
AfterEnd	Immediately after the closing tag and before any other content
BeforeBegin	Immediately before the opening tag
BeforeEnd	Immediately before the closing tag

Warnings:

- You cannot call this method while the document is loading.
- MSIE version 4 for Macintosh does not support this as widely as the Windows version or later versions of MSIE for Macintosh.

See also:	Element object, Element.getAdjacentText(), Element.innerHTML, Element.innerText, Element.insertAdjacentText(), Element.outerHTML, Element.outerText

Element.insertAdjacentText() (Method)

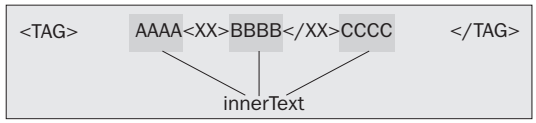
Inserts some text into the document adjacent to this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	IE	*`myElement`*`.insertAdjacentHTML(`*`aRelativePosition, aString`*`)`
Argument list:	*`aRelativePosition`*	An indication of where the new HTML is to be placed
	`aString`	The new fragment of text to be inserted

This is quite similar to an expression that assigns a new value to the `innerText` property of an `Element` object. In this case, the new text leaves the existing text and HTML content intact.

The relative positions where the adjacent text can be introduced determine where the new text is to be inserted. The following relative positions can be specified:

Position:	Description:
`AfterBegin`	Immediately after the opening tag and before any other content
`AfterEnd`	Immediately after the closing tag and before any other content
`BeforeBegin`	Immediately before the opening tag
`BeforeEnd`	Immediately before the closing tag



Warnings:

- You cannot call this method while the document is loading.
- MSIE version 4 for Macintosh does not support this as widely as the Windows version or later versions of MSIE for Macintosh.

See also:	`Element` object, `Element.getAdjacentText()`, `Element.insertAdjacentHTML()`, `Element.replaceAdjacentText()`,`Node.insertBefore()`

Element.isContentEditable (Property)

Returns a Boolean value indicating whether the content of the element can be altered.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myElement`*`.isContentEditable`

It is necessary to have this property because the `Element.contentEditable` property may have the value "inherited" assigned to it. To determine whether the content was in fact editable would require you to walk up the document hierarchy to a parent node whose `contentEditable` property was set explicitly.

Property attributes:

`ReadOnly`.

Element.isDisabled (Property)

A flag property that indicates whether the user can interact with the object or not.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myElement`*`.isDisabled`

This is especially applicable to `input` objects or those other objects that may not be sub-classes of the `Input` object but can be clicked on by the user. For example, `Anchor` objects.

The property yields the current state of the disabled setting for the receiving object.

A separate property to the object's `disabled` property is necessary because it might depend on the context in which an object is used and whether that context (such as a form or window) is disabled. The implication is then that all child objects within that context would be disabled too. It's also necessary because some objects are not members of the `Input` object family but can still be disabled even though they do not have a `disabled` property of their own.

Property attributes:

`ReadOnly`.

Element.isTextEdit (Property)

A Boolean indication of the text editing capabilities of this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myElement`*`.isTextEdit`

This property contains a Boolean flag to indicate whether the object can be used to create a `TextRange` object. This is done with the `createTextRange()` method. Only the following object classes should yield a true value for this property:

- `BODY`
- `BUTTON`
- `TextCell`
- `TextArea`

If the HTML element represents an `<INPUT>` tag whose type is `TEXT` for example, then this value will yield true. Most of the time it will yield false. It is useful for writing generic accessors and handlers for objects which may or may not have selectable or editable text in them.

Logically you would only want to read from this property.

The value yielded by this property will be true if the element represents an editable text field and false if is not.

Example code:

```
// An example showing how a text range can be created//
conditionallyif(myElement.isTextEdit){   myElement.createTextRange();}
```

See also:	// An example showing how a text range can be created
	// conditionally
	if(myElement.isTextEdit)
	{
	  myElement.createTextRange();
	  alert("TextRange object created");
	}

Property attributes:

`ReadOnly`.

Element.lang (Property)

The value of the `LANG` tag attribute of the HTML tag that represents this element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myElement*`.lang`
	-	*myElement*`.lang = `*aNationality*
Argument list:	*aNationality*	An indication of the national language required

This property controls the locale specific text rendering. It allows special characters to be handled appropriately and special character sets to be supported properly according to the national language variants.

Refer to the Language codes topic for a list of the available language codes.

See also:	`Element` object, `LANG="..."`, `Navigator.language`, `Navigator.systemLanguage`, `Navigator.userLanguage`

Cross-references:

RFC 1766

http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt

http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

Element.language (Property)

The required scripting language for fragments of script associated with this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myElement*`.language`
	IE	*myElement*`.language = `*aScriptLanguage*
HTML syntax:	`<SCRIPT LANGUAGE="...">`	
Argument list:	*aScriptLanguage*	One of the available scripting languages

This does not affect scripting contained within script tags, but provides a way to select a script language interpreter for use with event handlers for example.

The following scripting languages are supported in MSIE (Netscape does not support this facility):

- JAVASCRIPT
- JSCRIPT
- VBS
- VBSCRIPT

The values are case-insensitive in the MSIE browser.

With this property available, it is conceivable that other languages could be supported in the future. For example, client-side Perl and Python are feasible. Search out the PerlScript and ActivePython projects for further details.

See also:	`<SCRIPT LANGUAGE="...">`, `Element` object

Element.lastChild (Property)

The last child object contained within the DOM hierarchy that descends from this element.

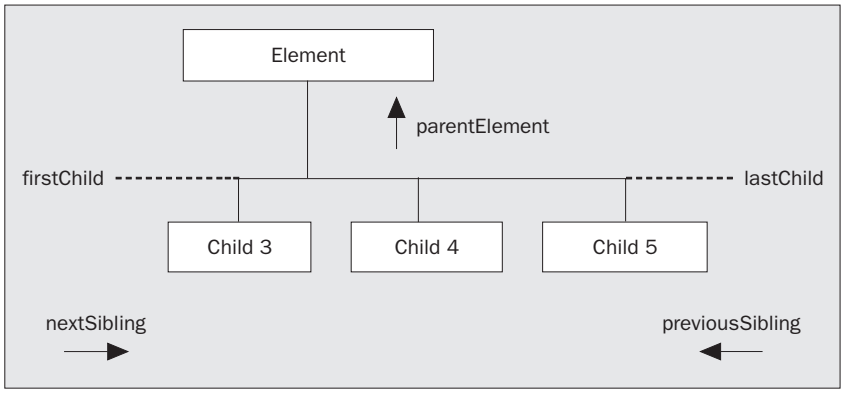
Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`Element` object
JavaScript syntax:	- *myElement*`.children[`*anIndex*`]` - *myElement*`.lastChild`
Argument list:	*anIndex* — A value equal to the length of the array minus 1

The `Element` objects are instantiated as the HTML tags are parsed within the document source. They are added to several collections and can be navigated in a variety of ways.

Each `Element` object has an array of `Element` objects considered to be its direct descendants (that is children). This collection will not contain its children's children. The last element in this collection can be referred to by index and its siblings by means of an enumerator or `for( ... in ... )` loop. However you can access it directly with this property.

Accessing the object at the end of the collection with this property is more convenient than measuring the length of the collection and accessing by array index.

If the element has no children, then this property will contain a `null` value.



See also:	`Element` object, `Element.firstChild`, `Element.nextSibling`, `Element.parentElement`, `Element.previousSibling`, `Node.lastChild`

Element.mergeAttributes() (Method)

Merge the attributes from one object with another.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`undefined`	
JavaScript syntax:	IE	*`myElement`*`.mergeAttributes(`*`anElement`*`)`
	IE	*`myElement`*`.mergeAttributes(`*`anElement`*`, `*`aCaseSense`*`)`
Argument list:	*`aCaseSense`*	A flag indicating whether name lookup is case sensitive
	`anElement`	A source element to merge attributes from

The attributes from the source object are merged with the attributes of the receiving object. This may replace some attribute values or may add to the set of attributes if similarly named attributes are present in both but only differ in their case sensitivity.

See also:	`Element.attributes[]`, `Element.removeAttribute()`, `Element.setAttribute()`

Element.nextSibling (Property)

An HTML element at the same level within the document hierarchy.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

Property/method value type:	`Element` object	
JavaScript syntax:	-	*myElement*.nextSibling
	-	*myElement*.parentElement.children[*anIndex*]
Argument list:	*anIndex*	An enumerated position in the collection

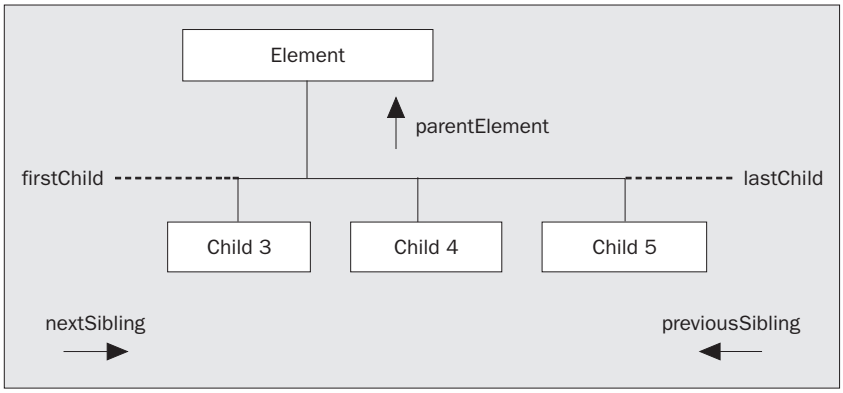
The `Element` objects are instantiated as the HTML tags are parsed within the document source. They are added to several collections and can be navigated in a variety of ways.

Each `Element` object has an array of `Element` objects considered to be its direct descendants (that is children). This collection will not contain its children's children. All elements in this collection can be referred to by index and all siblings by means of an enumerator or `for( ... in ... )` loop. However you can access them directly with this property.

Accessing the next object in the collection belonging to the parent by means of this property is more convenient than traversing back to the parent and working through the children property's collection.

The next sibling will be the object following this one in the children property's collection of the same parent element.

If there is no next sibling then the `null` value will be returned.



See also:	`Element` object, `Element.firstChild`, `Element.lastChild`, `Element.parentElement`, `Element.previousSibling`, `Node.nextSibling`

Element.nodeName (Property)

Part of the internal document hierarchy management.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myElement*.nodeName

The nodeName property is part of the browser's internal DOM management. It contains a value that is the name of the HTML tag that instantiated the object. This is also reflected as the tagName property.

When the object is a Text object, the nodeName value is "#text" because there is no HTML tag to represent plain text and a text node is placed between HTML Element objects. It's like the mortar between the bricks in a wall.

See also:	Element object, Node.nodeName

Property attributes:

ReadOnly.

Element.nodeType (Property)

Part of the internal document hierarchy management.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myElement*.nodeType

The nodeType property is part of the browser's internal DOM management.

By inspecting the properties of various objects with an enumeration loop, you can determine some values for this property. If you know the node type, you may be able to take advantage of this when you write more sophisticated scripts. Here is a partial list of node types defined by the DOM specification:

Constant:	Type:	Description:
undefined	null	A member of the attributes collection
ELEMENT_NODE	1	HTML element object node
ATTRIBUTE_NODE	2	HTML tag attribute object
TEXT_NODE	3	Text object node
CDATA_SECTION_NODE	4	CDATA section
ENTITY_REFERENCE_NODE	5	Entity reference
ENTITY_NODE	6	Entity node

Table continued on following page

Constant:	Type:	Description:
PROCESSING_INSTRUCTION_NODE	7	Processing instruction node
COMMENT_NODE	8	Comment node
DOCUMENT_NODE	9	Document object
DOCUMENT_TYPE_NODE	10	Doctype object
DOCUMENT_FRAGMENT_NODE	11	Document fragment node
NOTATION_NODE	12	Notation node

The DOM standard is quite large at level 1 and defines a lot of ways in which you can traverse a document structure. The level 2 capabilities are even more extensive. Scripts than can understand node types and DOM document trees will be able to accomplish some amazingly powerful things.

See also:	`Element` object, `Node.nodeType`

Property attributes:

`ReadOnly`.

Element.nodeValue (Property)

Part of the internal document hierarchy management.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *`myElement`*`.nodeValue`

This object is part of the browser's internal DOM management.

The value of this attribute always seems to be the `null` object for objects that represent HTML tags. This is probably because all the other attributes of the tags can be stored in named properties of the `Element` object or in attributes of those properties collected into the `Attributes` object that belongs to it.

When the node is a `#text` value, the text itself is stored here.

See also:	`Element` object, `Node.nodeValue`

Element.normalize() (Method)

Processes all of the `textNode` objects that are children of the receiving element and prepares the document for save and restore.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	`Element.normalize()`

There are implications here for the way `CDATASection` objects are handled and are most likely to be used when XPointer lookups are required.

This begins to move in the direction of XML support, which the DOM actively facilitates and which is a very large and complicated topic. For now we shall concentrate on HTML documents.

Element.offsetHeight (Property)

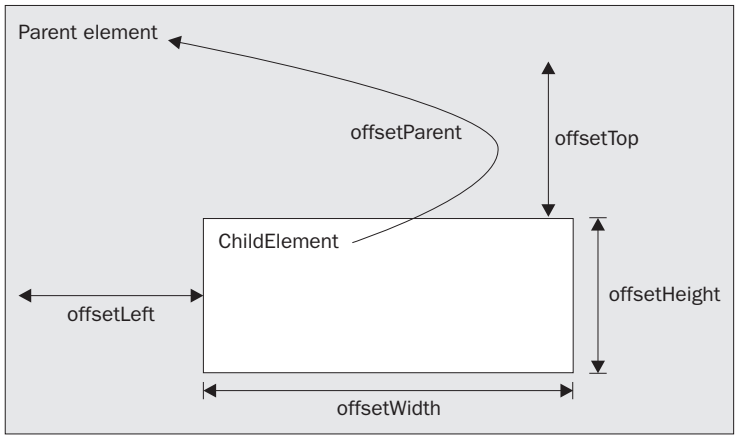
The height of this element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.offsetHeight`

The height of the element in pixels. This value is read-only but for some objects such as images the `height` property should be used instead if you need to modify the physical size of the item. The changes will be reflected in this property but you cannot make the changes by altering the value of this property from your script.

The value in this property is not supposed to include any padding, border or margin widths.

Although the name of this property implies its value is related to a parent object, in fact it is a measure of the element's height and the parentage has no bearing on that whatsoever.



Warnings:

- The offset properties are implemented very inconsistently across platforms and versions of MSIE. Use them with caution. In any case, they are available for read access only so they are of limited use in position control.
- MSIE 4 for Windows incorrectly includes padding widths around the object when calculating the size value in here. The correct value will be yielded on both Macintosh and Windows versions of MSIE 4 if you don't specify any padding at all.

See also:	`Element` object, `Element.offsetWidth`

Property attributes:

`ReadOnly`.

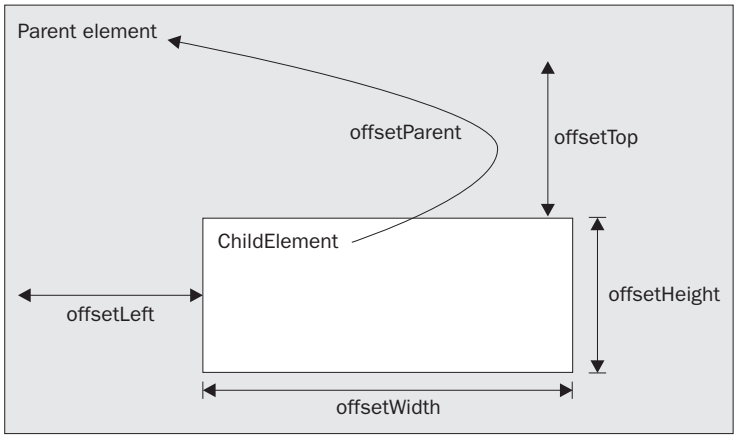
Element.offsetLeft (Property)

The X coordinate of the HTML element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.offsetLeft`

The position of the object in pixels offset horizontally from its parent. This value is read-only.

The `offsetParent` property contains a reference to the parent object that the relative positions are measured with respect to.



Warnings:

- The offset properties are implemented vary inconsistently across platforms and versions of MSIE. Use them with caution. In any case, they are available for read access only so they are of limited use in position control.

See also:	`Anchor.x`, `Element` object, `Element.offsetParent`, Measurement units, `style.pixelLeft`, `style.posLeft`

Property attributes:

`ReadOnly`.

Element.offsetParent (Property)

A reference to the parent object that is the reference point for offset positioning values.

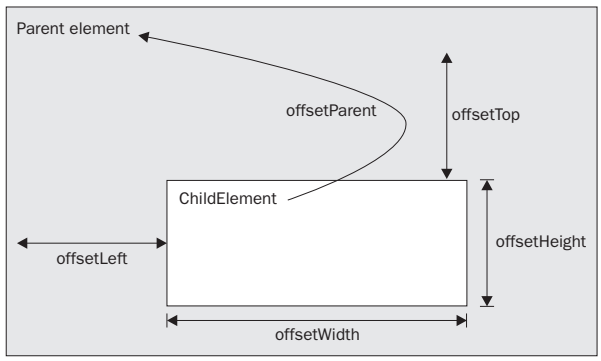
Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Element` object	
JavaScript syntax:	IE	*myElement*`.offsetParent`

The parent object whose position provides a datum for the element to be offset away from.

This could refer to the `document` object for the current window. In many cases it will be the `BODY` object for the document. However you may have used `<TABLE>` tags to control the layout and these might be the parent objects for the content of the table while the parent of the `TABLE` objects would be the `BODY` object.

The parent-child hierarchy of `element` objects within the document is dependent on how you construct it when the document source is authored.

This is not the same as the `parentElement` property. That property reflects the true document hierarchy in all its fine detail. The `offsetParent` property reflects the spatial relationships between objects. Objects that show up in the `parentElement` hierarchy will likely be omitted from the `offsetParent` hierarchy model on account of them not contributing any spatial offset to the contained elements.



Warnings:

- The offset properties are implemented very inconsistently across platforms and versions of MSIE. Use them with caution. In any case, they are available for read access only so they are of limited use in position control.

See also:	`BODY` object, `Document` object, `Element` object, `Element.offsetLeft`, `Element.offsetTop`, `Element.parentElement`, Hierarchy of objects, `TABLE` object

Property attributes:

`ReadOnly`.

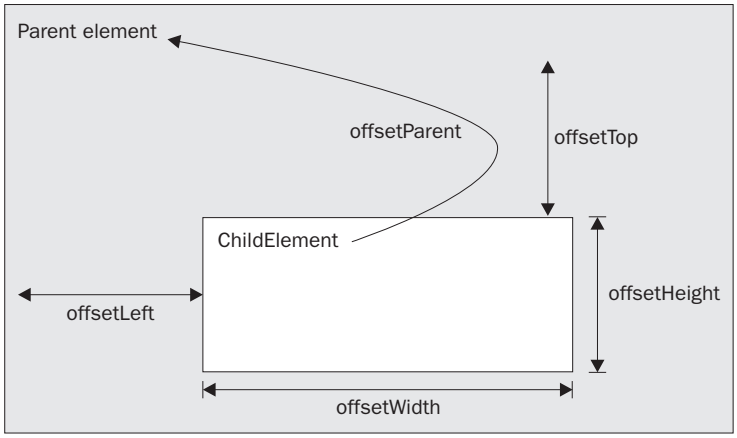
Element.offsetTop (Property)

The Y coordinate of the HTML element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.offsetTop`

An offset value in the vertical access from the origin of the parent to the top of the receiving `Element` object. This value is read-only.

The `offsetParent` property contains a reference to the parent object that the relative positions are measured with respect to.



Warnings:

- The offset properties are implemented very inconsistently across platforms and versions of MSIE. Use them with caution. In any case, they are available for read access only so they are of limited use in position control.

See also:	`Anchor.y`, `Element` object, `Element.offsetParent`, Measurement units, `style.pixelTop`, `style.posTop`

Property attributes:

`ReadOnly`.

Element.offsetWidth (Property)

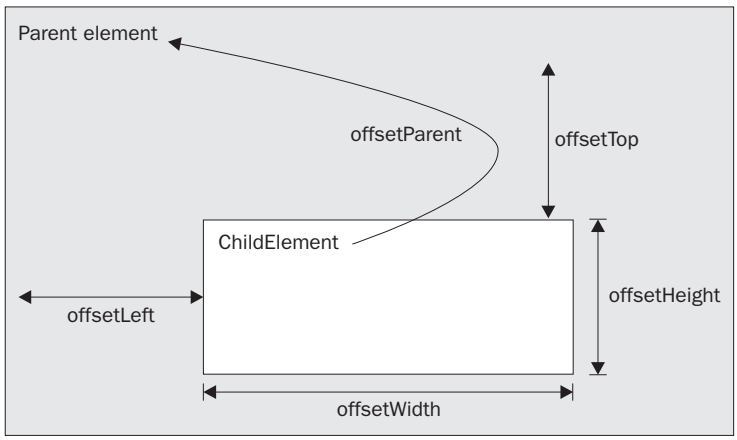
The width of the HTML element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.offsetWidth`

The width of the element in pixels. This value is read-only, but for some objects such as images the `width` property should be used instead if you need to modify the physical size of the item. The changes will be reflected in this property but you cannot make the changes by altering the value of this property from your script.

The value in this property is not supposed to include any padding, border or margin widths.

Although the name of this property implies its value is related to a parent object, in fact it is a measure of the element's width and the parentage has no bearing on that whatsoever.



Warnings:

- The offset properties are implemented very inconsistently across platforms and versions of MSIE. Use them with caution. In any case, they are available for read access only so they are of limited use in position control.
- MSIE version 4 for Windows incorrectly includes padding widths around the object when calculating the size value in here. The correct value will be yielded on both Macintosh and Windows versions of MSIE 4 if you don't specify any padding at all.

See also:	`Element` object, `Element.offsetHeight`

Property attributes:

`ReadOnly`.

Element.onevent (Property)

A property containing a reference to an event handler property.

Availability:	JavaScript – 1.2 JScript – 5.0 Internet Explorer – 5.0 Netscape – 4.0	
Property/method value type:	`Function` object	
JavaScript syntax:	- *myElement*`.onevent`	
	- *myElement*`.onevent = `*aFunctionObject*	
HTML syntax:	`< ... onEvent="...">`	
Argument list:	*aFunctionObject*	An event handler function

There is a small set of event handler function properties created by default. Some sub-classes of the `Element` object will add others as needed. You can assign your own handlers to these properties (if necessary creating the properties that don't already exist).

Here is a list of all the event handler property names discovered during our research. These are collated from a variety of documentation sources. Some were discovered by inspecting objects with scripts:

Event:	Handler:	Usage:
Abort	`onabort`	When image loading is aborted.
AfterPrint	`onafterprint`	When printing has just finished.
AfterUpdate	`onafterupdate`	When an update is completed.
Back	`onback`	The user has clicked on the [BACK] button in the toolbar.
BeforeCopy	`onbeforecopy`	Immediately before a copy to the clipboard.
BeforeCut	`onbeforecut`	Immediately before a cut to the clipboard.

Table continued on following page

Event:	Handler:	Usage:
BeforeEditFocus	`onbeforeeditfocus`	Immediately before the edit focus is directed to an element.
BeforePaste	`onbeforepaste`	Immediately before the clipboard is pasted.
BeforePrint	`onbeforeprint`	Immediately before printing begins.
BeforeUnload	`onbeforeunload`	Called immediately prior to the window being unloaded.
BeforeUpdate	`onbeforeupdate`	Called immediately before an update commences.
Blur	`onblur`	When an input element loses input focus.
Bounce	`onbounce`	Triggered when a marquee element hits the edge of its element area.
Change	`onchange`	When edit fields have new values entered or a popup has a new selection, this event's handler can check the new value.
Click	`onclick`	When the user clicks the mouse button on the `Element` object that represents the object on screen.
ContextMenu	`oncontextmenu`	Special handling for contextual menus.
Copy	`oncopy`	When a copy operation happens.
Cut	`oncut`	When a cut operation happens.
DataAvailable	`ondataavailable`	Some data has arrived asynchronously from an applet or data source.
DataSetChanged	`ondatasetchanged`	A data source has changed the content or some initial data is now ready for collection.
DataSetComplete	`ondatasetcomplete`	There is no more data to be transmitted from the data source.
DblClick	`ondblclick`	When the user double clicks on an object.
Drag	`ondrag`	When a drag operation happens.
DragDrop	`ondragdrop`	Some data has been dropped onto a window.
DragEnd	`ondragend`	When a drag ends.
DragEnter	`ondragenter`	When a dragged item enters the element.
DragLeave	`ondragleave`	When a dragged item leaves the element.
DragOver	`ondragover`	While the dragged item is over the element.
DragStart	`ondragstart`	The user has commenced some data selection with a mouse drag.
Drop	`ondrop`	When a dragged item is dropped.
Error	`onerror`	Triggered if an error occurs when loading an image.
ErrorUpdate	`onerrorupdate`	An error has occurred in the transfer of some data from a data source.
FilterChange	`onfilterchange`	A filter has changed the state of an element or a transition has just been completed.
Finish	`onfinish`	A marquee object has finished looping.
Focus	`onfocus`	When the form element is selected for entry.

Table continued on following page

Event:	Handler:	Usage:
Forward	`onforward`	The user has clicked on the [FORWARD] button in the toolbar.
Help	`onhelp`	The user has pressed the [F1] key or selected [help] from the toolbar or menu.
KeyDown	`onkeydown`	When a key is pressed.
KeyPress	`onkeypress`	Pressing the key down and releasing it again elicits this event.
KeyUp	`onkeyup`	When a key is released.
Load	`onload`	When an object has completed loading.
LoseCapture	`onlosecapture`	When an element loses event capturing permission.
MouseDown	`onmousedown`	When the mouse button is pressed.
MouseDrag	`onmousedrag`	An event handler for mouse drag operations.
MouseMove	`onmousemove`	When the mouse pointer is moved.
MouseOut	`onmouseout`	When the mouse pointer leaves the active area occupied by the `Element` object that represents the object on screen.
MouseOver	`onmouseover`	When the mouse pointer enters the active area owned by the object.
MouseUp	`onmouseup`	When the mouse button is released.
Move	`onmove`	The browser window has been moved.
Paste	`onpaste`	When a paste operation happens.
PropertyChange	`onpropertychange`	When an object property is modified (similar to the Netscape Navigator `watch()` method).
ReadyStateChange	`onreadystatechange`	An object in the window has changed its ready state.
Reset	`onreset`	The user has clicked a reset button in a form.
Resize	`onresize`	As the window is resized, this event is triggered.
RowEnter	`onrowenter`	The data in a field bound to a data source is about to be changed
RowExit	`onrowexit`	The data in a field bound to a data source has been changed.
Scroll	`onscroll`	The window has been scrolled.
Select	`onselect`	Some textual content in the window has been selected.
SelectStart	`onselectstart`	A select action is beginning.
Start	`onstart`	A marquee element is beginning its loop.
Stop	`onstop`	When a stop action occurs.
Submit	`onsubmit`	The user has clicked on the submit button in a form.
Unload	`onunload`	Triggered when the document is unloaded.

You can usually discover what events an object supports by enumerating all its properties in a `for( ... in ... )` loop. The event handler hooks will be listed with the other enumerable properties of the object.

Element.outerHTML (Property)

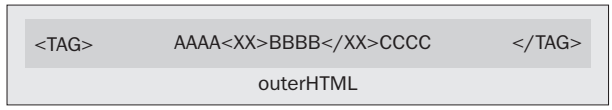
The outer HTML of a tag in the document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myElement*`.outerHTML`
	IE	*myElement*`.outerHTML = `*someHTML*
Argument list:	*someHTML*	Some new HTML content to place outside the object

The outer HTML of an HTML element is that fragment of HTML that completely contains the start and end tags of the element. The `outerHTML` of an `<A>` tag would be the complete anchor and any styled text or image tags between the `<A>` and `</A>` tags.

You can read this value to extract a fragment of HTML and you can write to this value to redefine a section of HTML within the page.

Because this property is not DOM compliant, you may want to refer to the DOM `Text` object and DOM `CharacterData` object which provide a compliant set of accessors to the content of a DOM component. The implementation of this is still somewhat vague and ambiguous and some more work needs to be done to bring the DOM compliant capabilities up to the same level of functionality. This can probably best be simulated by walking up the tree to a containing element and using `innerHTML` on that object.



Warnings:

- Be careful if you extract the `outerHTML` of an element and `document.write()` it back to the same document. You can create recursive loop situations if you are evaluating in global code during the document loading process.
- You cannot set this property while the document is loading.
- If you replace the outer HTML, you will also be replacing the containing tags of the object. If you do not keep the `ID` or `NAME` properties intact, you may not be able to locate the object again in the way you might expect to later.
- Conceptually, changing the `outerHTML` property of an object might imply that the object's class is being changed as well. This is not truly consistent with the rules of object oriented programming as you would not normally 'cast' objects in the way you might 'cast' pointers in the C-Language context. The effects of this are not documented and you may find that the behavior varies from one implementation to another.

- MSIE version 4 for Macintosh does not support this as widely as the Windows version or later versions of MSIE for Macintosh.
- Netscape does not support it at all.

See also:	`Element` object, `Element.innerHTML`, `Element.innerText`, `Element.insertAdjacentHTML()`, `Element.outerText`

Element.outerText (Property)

The outer text of a tag in the document.

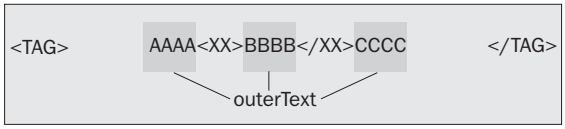
Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myElement`*`.outerText`
	IE	*`myElement`*`.outerText = `*`aString`
Argument list:	*`aString`*	Some new text content to place outside the object

The HTML from the start tag to the end tag, including the tags that define the element, is extracted and any tags are removed. The resulting text is returned when this property is being read.

You can read this value to extract a fragment of text, and you can write to this value to redefine a section of text within the page.

When the property is being written to, the start and end tags are destroyed because everything between them and the tags is replaced with a text string. To place new HTML tags as replacements for the ones you are about to destroy, use the `outerHTML` property.

DOM standardization provides access to the content of a `Text` and `CharacterData` object. These are special cases of the `Node` object which is also the parent of the `Element` object. We really need the `Element` object to inherit the data accessors of the `CharacterData` object as well and for now at least it doesn't. So long as you want to access text in between `Element` objects, you may be able to navigate the Node hierarchy and access properties of the `Text` objects between each `Element`.



Warnings:

- Be careful if you extract the `outerText` of an element and `document.write()` it back to the same document. You can create recursive loop situations if you are evaluating in global code during the document loading process.
- You cannot set this property while the document is loading.

- It is not clear what the intent of this property is when used as an `LValue`. Assigning a new string to this property when it's on the left of an expression implies that the containing HTML tags could be removed. Some experimentation may be necessary to ensure you get predictable behavior across the various browser implementations. Removing the containing HTML tags could cause an object to be removed from the document tree and discarded. Because the container is no longer there, any child objects will also have been destroyed and there is no handle by which you can locate the object again because there is no HTML tag with a `NAME` or `ID` attribute to find.
- If the implementation is smart, it will let you retain the parent object by reference, but will mutate it from an HTML Element that represents a tag and change it into a text node. That way you could then reassign some `outerHTML` to it to re-establish the HTML containment again.
- MSIE version 4 for Macintosh does not support this as widely as the Windows version or later versions of MSIE for Macintosh.
- Netscape does not support it at all.

See also:	`Element` object, `Element.innerHTML`, `Element.innerText`, `Element.insertAdjacentHTML()`, `Element.outerHTML`

Element.ownerDocument (Property)

A reference to the `document` object that the element is contained within.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`Document` object
JavaScript syntax:	- *myElement*`.ownerDocument`

This is another name for the `document` property of an `Element` object. They both refer to the same object.

This is provided by MSIE 5 and Netscape 6.0 as a convenience property. Accessing a containing document object in this way is far easier than walking down a `document.childElement` sequence to locate a contained document.

See also:	`Document` object, `Element` object, `Element.document`, Hierarchy of objects, `Node.ownerDocument`

Property attributes:

`ReadOnly`.

Element.parentElement (Property)

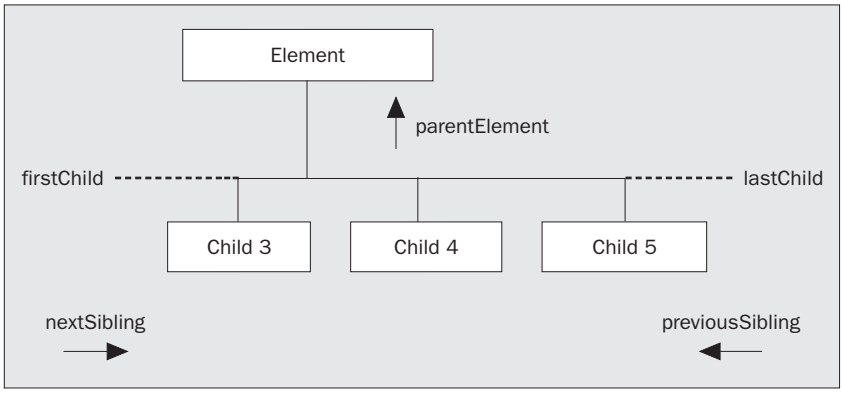
The element that is the owner or parent of the current object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Element` object	
JavaScript syntax:	IE	*`myElement`*`.parentElement`

This is the object that contains the receiving `Element` object. For example, an `Element` object that represents a `<TABLE>` tag is the parentElement of the `Element` objects that represent the `<TR>` tags within that table. If you want to add a row to a table, you may select the row that needs to be replaced or that it can be inserted after. You can then locate its `parentElement` and modify that if necessary.

This is not the same as the `offsetParent`. This property reflects the true document hierarchy in all its fine detail. The `offsetParent` property reflects the spatial relationships between objects. Objects that show up in the `parentElement` hierarchy will likely be omitted from the `offsetParent` hierarchy model on account of them not contributing any spatial offset to the contained elements.

Access to a `parentElement` is especially useful when driving the generation of `styleSheet` properties. You might have a series of nested containers which cascade a relative font size. By checking the fonts size property belonging to the `style` object associated with the `parentElement`, you can set a limit on this cascading effect to prevent the font sizes from becoming too small.



See also:	`Document` object, `Element` object, `Element.firstChild`, `Element.lastChild`, `Element.nextSibling`, `Element.offsetParent`, `Element.parentNode`, `Element.previousSibling`, Hierarchy of objects

Property attributes:

`ReadOnly`.

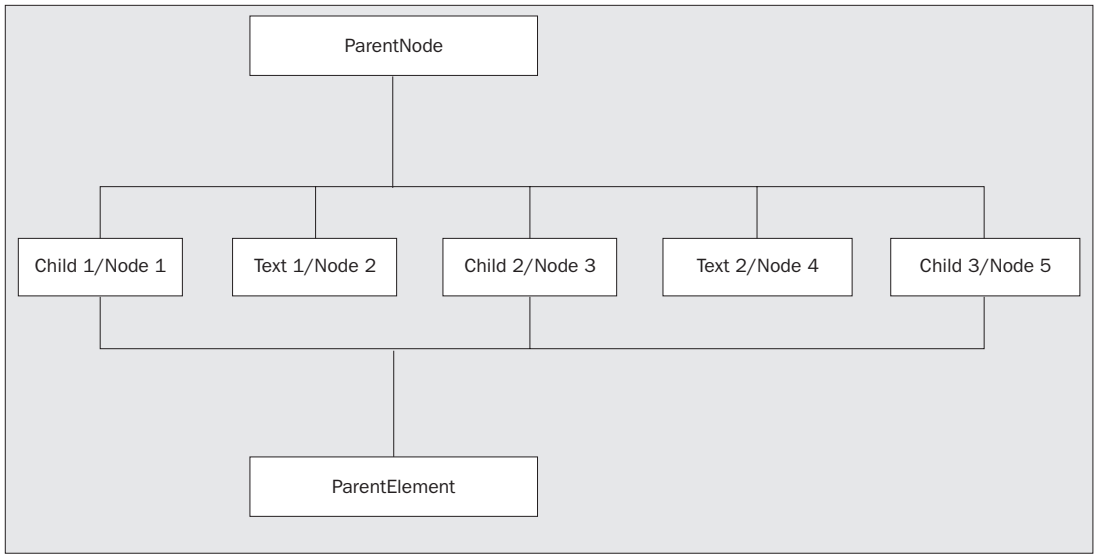
Element.parentNode (Property)

Part of the internal DOM hierarchy management.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`Element` object	
JavaScript syntax:	-	*`myElement`*`.parentNode`

This is part of the internal DOM management provided by the browser. Although the functionality is similar to `parentElement`, it is not quite the same since there are two hierarchies superimposed on one another. One hierarchy simply contains objects representing the HTML tags. The other contains additional objects that help the browser maintain the text between the HTML tags.

The `parentElement` property is not DOM compliant whereas `parentNode` is.



See also:	`Document` object, `Element` object, `Element.parentElement`, Hierarchy of objects, `Node.parentNode`

Property attributes:

`ReadOnly`.

Element.parentTextEdit (Property)

A reference to the next highest object in the hierarchy that allows a `TextRange` to be created.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Element` object	
JavaScript syntax:	IE	*`myElement`*`.parentTextEdit`

`TextRange` objects can only be created for a limited sub-set of the available objects in the MSIE browser. Only objects that can receive input focus are appropriate.

An object somewhere higher up in the hierarchy is referred to by this property. The document hierarchy may have to be traversed upwards for some considerable distance before a suitable object is found, so this hierarchical model is much more sparsely populated than any of the others.

Warnings:

- Because text ranges are not supported in the Macintosh, MSIE returns a `null` value when text ranges and the `parentTextEdit` property is accessed.

Example code:

```
// Locate a textEdit capable parent object and create an all enclosing text range.
myParentTextEdit = myElement.parentTextEdit;
myTextRange = myParentTextEdit.createTextRange();
alert ("TextRange object created for " + myParentTextEdit.tagName);
```

See also:	`Element` object, Hierarchy of objects, `TextRange` object

Property attributes:

`ReadOnly`.

Element.previousSibling (Property)

An HTML element at the same level within the document hierarchy.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Element` object	
JavaScript syntax:	-	*`myElement`*`.parentElement.children[`*`anIndex`*`]`
	-	*`myElement`*`.previousSibling`

Argument list:	*anIndex*	An enumerated position in the collection

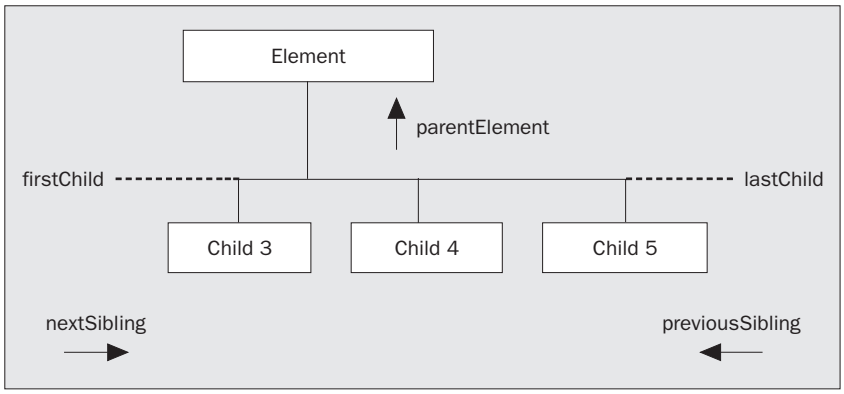
The `Element` objects are instantiated as the HTML tags are parsed within the document source. They are added to several collections and can be navigated in a variety of ways.

Each `Element` object has an array of `Element` objects that are considered to be its direct descendants (that is children). This collection will not contain its children's children. All elements in this collection can be referred to by index and all siblings by means of an enumerator or `for( ... in ... )` loop. However you can access them directly with this property.

Accessing the next object in the collection belonging to the parent by means of this property is more convenient than traversing back to the parent and working through the children property's collection.

The previous sibling will be the object prior to this one in the children property's collection of the same parent element.

If there is no previous sibling, then the `null` value will be returned.



See also:	`Element` object, `Element.firstChild`, `Element.lastChild`, `Element.nextSibling`, `Element.parentElement`, `Node.previousSibling`

Element.readyState (Property)

The `readyState` of an object contains a value indicating whether it is currently loading or is ready to use.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myElement*`.readyState`

This property contains the current disposition of the `element` object.

You might use this in a timed event to watch the status of a document being loaded into a window or frame. Then when it has completed loading, you could activate some script within it.

Here is a list of the ready state values:

State:	Value:
uninitialized	The object is first instantiated, but has not begun loading.
loading	The object has commenced loading.
loaded	The object has completed loading.
interactive	The object is loaded but not yet closed. However, it is ready to handle interaction.
complete	The object body has been closed and the loading is finished.

Note that these are string values. Some references describe a numeric value for ready states. According to the Microsoft documentation however, they are strings.

Sometimes, you can design scripts to execute while the document is downloading. Inline scripts for example. At that time, you may even be able to trigger interval timed deferred executions as well.

An object may not need to reflect the complete status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an `OBJECT` object that represents an `<OBJECT>` tag until the `<BODY>` has completed loading. This is because the `ActiveX` object construction requires a complete document body structure to attach itself to.

Every time this `readyState` value changes, it triggers an `onReadyStateChange` event call-back.

See also:	`onReadyStateChange`

Property attributes:

`ReadOnly`.

Element.releaseCapture() (Method)

Part of the event handling mechanism in MSIE.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myElement*`.releaseCapture()`

If event capture was established with the `setCapture()` method, this relinquishes such event capturing.

See also:	`Element.setCapture()`

Element.removeAttribute() (Method)

An accessor method to delete a named custom attribute.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myElement*`.removeAttribute(`*anAttribName*`)`
	-	*myElement*`.removeAttribute(`*anAttribName*`, `*aCaseSense*`)`
Argument list:	*aCaseSense*	A flag indicating whether lookup is case sensitive or not
	anAttribName	An attribute of an `Element` object

This is an accessor method which is used to remove named attributes of an `Element` object. Attributes are not properties in the strict sense of the word but may be accessible as if they were in some implementations. They are gathered together into an `Attributes` collection which you can access like any other `Array` or `Collection` object to retrieve the individual `Attribute` objects. The accessor methods provide some additional modes of access for convenience.

This accessor is intended to provide a means of managing custom attributes.

If the `false` value is returned, it may mean that either the attribute did not exist and could not be removed or that the attribute was locked in some way preventing it from being removed. The `true` value returned by this method indicates the attribute was successfully removed.

Note also that you can only remove attributes that were added with the `setAttribute()` method. Some attributes may have been added by the browser and may not be removed by scripts.

Warnings:

- Note that the DOM level 1 standard defines this method as returning no meaningful value. Other implementations may concur so you should not rely on the value being returned.
- However, Netscape 6.0 and MSIE 5.5 do return a meaningful value.

See also:	`Attribute` object, `Attributes` object, `Element` object, `Element.getAttribute()`, `Element.mergeAttributes()`, `Element.setAttribute()`

Element.removeAttributeNode() (Method)

Removes an attribute node from the element hierarchy. If the attribute has a default value, that will be used subsequently.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Attribute` object	
JavaScript syntax:	-	*`myElement`*`.removeAttributeNode(`*`anAttribute`*`)`
Argument list:	*`anAttribute`*	An `attribute` object to be removed

See also:	`Attribute` object

Element.removeBehavior() (Method)

Removes a behavior that was previously added to the object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*`myElement`*`.removeBehavior()`

Part of the MSIE behaviors control suite.

See also:	Behavior, `Element.addBehavior()`

Element.removeExpression() (Method)

Removes an expression from an object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*`myElement`*`.removeExpression()`

This is used for removing expressions from the `style` object that have previously been put there by the `setExpression()` method call.

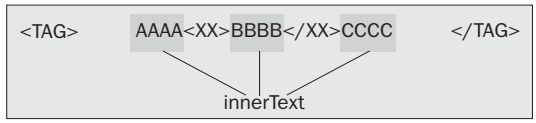
See also:	`Element.getExpression()`, `Element.setExpression()`, `style.getExpression()`, `style.removeExpression()`, `style.setExpression()`

Element.replaceAdjacentText() (Method)

Replace some text adjacent to the `element` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`undefined`	
JavaScript syntax:	IE	*`myElement`*`.replaceAdjacentText (`*`aRelativePosition`*`, `*`aString`*`)`
Argument list:	*`aRelativePosition`*	An indication of where the new HTML is to be placed
	`aString`	The new fragment of text to be inserted

This is a close relation to the `insertAdjacentText()` method. This method will replace the text adjacent to an object rather than add to it.



See also:	`Element.getAdjacentText()`, `Element.insertAdjacentText()`

Element.runtimeStyle (Property)

The style settings for the object, taking into account any cascaded styles and changes to the styles that may have happened dynamically.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Style` object	
JavaScript syntax:	IE	*`myElement`*`.runtimeStyle`

Because the style values are cascaded from style sheet to style sheet and may include some inline styles as well as some explicit styles, objects need to maintain a current style value that is the result of all the inheritances applied on top of one another.

In addition they maintain a runtime style which reflects dynamic changes as well. The runtime style is based on the current style in the first place.

The value of this property is a reference to a `style` object. However, it does not refer to the same `style` object as the `style` property. You can test for this with the `isObjectEqual()` function that we have documented elsewhere.

See also:	`currentStyle` object, `Element.currentStyle`, `Element.style`, `runtimeStyle` object

Element.scopeName (Property)

The name of a scope chain.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myElement`*`.scopeName`

The MSIE browser allows different scope chains to be built and referred to by name. These are called namespaces. This is a very esoteric property and its coverage is quite sparse in the Microsoft developer documentation. However, you should refer to the MSDN web site for further details.

Web-references:

http://msdn.microsoft.com/workshop/author/dhtml/reference/properties/scopename.asp

Element.scrollHeight (Property)

An MSIE property for measuring sizes of objects when they have been scrolled.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.scrollHeight`

This functionality is not well supported across platforms and means different things for different objects. Generally, it is taken as the height of an object when its scroll position is taken into account. Objects scrolled entirely off the screen have a `scrollHeight` of zero.

Warnings:

- This functionality is sufficiently unportable as to preclude its use from any useful purpose.

Property attributes:

`ReadOnly`.

Element.scrollIntoView() (Method)

Set the scroll of the document to bring the element into view.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	IE	*`myElement`*`.scrollIntoView()`
	IE	*`myElement`*`.scrollIntoView(`*`aReferencePoint`*`)`
Argument list:	*`aReferencePoint`*	Indicates the desired scroll to location

This method scrolls the window to bring the receiving element into view. The argument is an optional Boolean value that indicates whether the receiving `Element` object should be scrolled to the top (`true`) or the bottom (`false`) of the window.

By default, the value true is assumed if no value is specified.

See also:	`Element` object

Element.scrollLeft (Property)

An MSIE property for measuring position of objects when they have been scrolled.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.scrollLeft`

This functionality is not well supported across platforms and means different things for different objects. Generally, it is taken as the distance an object has been scrolled in the horizontal axis.

Warnings:

- This functionality is sufficiently unportable as to preclude its use from any useful purpose.

Element.scrollTop (Property)

An MSIE property for measuring position of objects when they have been scrolled.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.scrollTop`

This functionality is not well supported across platforms and means different things for different objects. Generally, it is taken as the distance an object has been scrolled in the vertical axis.

Warnings:

- This functionality is sufficiently unportable as to preclude its use from any useful purpose.

Element.scrollWidth (Property)

The width of a scrolling region.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myElement`*`.scrollWidth`

This functionality is not well supported across platforms and means different things for different objects. Generally, it is taken as the width of an object when its scroll position is taken into account. Objects scrolled entirely off the screen have a `scrollWidth` of zero.

Warnings:

- This functionality is sufficiently unportable as to preclude its use from any useful purpose.

Property attributes:

`ReadOnly`.

Element.setAttribute() (Method)

An accessor method to set a named custom attribute value.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*`myElement`*`.setAttribute(`*`anAttribName`*`,` *`aValue`*`)`
	-	*`myElement`*`.setAttribute(`*`anAttribName`*`,` *`aValue`*`,` *`aCaseSense`*`)`
Argument list:	*`aCaseSense`*	A flag indicating whether name lookup is case sensitive
	`anAttribName`	An attribute of an `Element` object
	`aValue`	A new value for the custom attribute

An attribute is added to the `attributes` collection for the object. Some attributes are presented as properties as well which is simply an alternative means of access to the same value. If that is the case then setting the attribute will also alter the property setting as well.

If the case sensitivity flag is set to true, then the name of the attribute must exactly match the name used in the HTML tag attribute, otherwise the results are uncertain.

See also:	`Element` object, `Element.getAttribute()`, `Element.mergeAttributes()`, `Element.removeAttribute()`

Element.setAttributeNode() (Method)

A new attribute node is added to the element. If one with the same name already exists, it will be replaced.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Attribute` object	
JavaScript syntax:	-	*myElement*`.setAttributeNode(`*anAttribute*`)`
Argument list:	*anAttribute*	An attribute node object to be set

See also:	`Attribute` object

Element.setCapture() (Method)

Part of the event handling mechanism in MSIE.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myElement*`.setCapture()`

If an event capture is established with this method, it can be relinquished with the `releaseCapture()` method.

See also:	`Element.releaseCapture()`

Element.setExpression() (Method)

Part of the behavior handling mechanisms in MSIE.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myElement*.setExpression(*aProperty*, *anExpression*, *aLanguage*)
Argument list:	*aProperty*	The name of a property to be modified
	anExpression	An expression to assign
	aLanguage	The language the expression is defined in

The rules that are used to construct a style are comprised of multiple expressions. You can use this method to assign a new value to an expression within a style item with that value being generated by a callback to a script function.

The rule might contain a line such as:

```
width:200px
```

The `setExpression()` method can be applied to the `style` object containing the rule with that expression like this:

```
myStyle.setExpression("width:", "callBack()", "JavaScript")
```

The value of the width parameter would be defined by the result of calling the `callBack()` function in the JavaScript context.

The language argument can be one of the following:

- JavaScript
- JScript
- VBScript

See also:	`Element.getExpression()`, `Element.removeExpression()`,`style.getExpression()`, `style.removeExpression()`,`style.setExpression()`

Element.sourceIndex (Property)

The index value of this element in the `document.all[]` array.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myElement*.sourceIndex

In MSIE, all `Element` objects are available in the `all[]` array object for the document. If you need to know the element's position within that order, this property tells you what the index value for the object is. This will change as objects are created and destroyed in the page.

This is useful to be able to tell the relative position of objects in the page, perhaps so that you can scroll forwards or backwards to reach them.

Warnings:

- This value may change if new objects are added to the document.

See also:	`Element` object

Property attributes:

`ReadOnly`.

Element.style (Property)

A `style` object that can be modified for the element.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0
Property/method value type:	`Style` object
JavaScript syntax:	- *`myElement`*`.style`

This property is an important reference to a DOM level 2 `style` object which can be accessed from script to dynamically alter the appearance of an HTML element.

The `Element.style` property yields an object that contains the style settings for the HTML element. By modifying the properties of this `style` object belonging to the HTML element, you can affect the display of the element. It is as if the element owned a miniature style sheet of its own. This may be affected by the settings of the `CLASS` tag attribute on the HTML tag that instantiates the `Element` object; we will come back to that in a little while.

You can replace the `style` object with another you have created separately or defined in the style sheet, and by doing this you can effect a considerable change in appearance of an object in one hit without needing to change several properties individually.

Style management is quite complex and offers many more capabilities than using HTML tags and attributes. You should be able to completely abstract any appearance related controls away from the HTML document at the cost of it not working on older browsers.

The complexity stems from there being an inheritance mechanism and the possibility of having many `style` objects attached to a document at different points. There is a hierarchy of `style` objects, each overriding another. Ultimately the appearance you see is the accumulation of all of those style controls cascaded down to your element.

You can try to alter a style property on an object only to see it remain the same as it was before. Styles are attached to objects at different points in the DOM tree and some style properties are inherited. So you could be altering a style property that is being masked. This can be quite frustrating.

Each HTML element in the DOM structure has a `style` property that points at a `style` object. `Style` objects may be shared by referring to the same one from several elements.

The `style` object is documented extensively elsewhere with a topic describing each of its properties. For now, let's look at some simple style manipulation. You can access the style properties of an element as simply as this:

```
myElement.style.color = "red";
```

That's it, aside from learning about all of the different style properties that are available and their values. You'll need to look at how styles are inherited down the DOM document tree.

Styles can be put into style sheets and pulled in from a shared document in a `<LINK>` tag. Access to those `style` objects is a little more complicated because you need to locate the object representing that `<LINK>` tag, then navigate through the `StyleSheet` object to find the rules that describe each named style block. Those rules are associated with your element by means of the `CLASS` HTML tag attribute. Having located the rule for your `element` object (and these would be shared between many objects), you can examine the rule object and access it's `style` property. That points at a `style` object that is just like the one your `Element.style` property would have pointed at.

You can also specify the styles inline, that is you can add a `STYLE` HTML tag attribute to the tag that instantiates your `element` object. That would be reflected in the properties of a private `style` object that your `Element.style` property refers to.

MSIE supports two additional properties that relate to the cascaded style on an element. They are called `currentStyle` and `runtimeStyle`. The `currentStyle` property points at an object that holds the accumulated style, taking into account linked stylesheets, tag attributes and anything else that can affect style appearance apart from any script driven changes. The `runtimeStyle` property takes the `currentStyle` property and accumulates appearance changes when styles are modified on an object, so it represents the visual style as it is now. Neither of these are portable though, and they are not part of the DOM standard and so they should be avoided.

To learn more about styles and how they work you might want to go through this learning sequence:

- Build a document with HTML tag driven appearance, just like you always used to.
- Now remove all its appearance control and introduce `STYLE` HTML tag attributes and put in some CSS style control.
- If you prefer, use a `<LINK>` tag to call in a style sheet and build some even more complex style controls.

- Now explore the document hierarchy by looking at individual HTML Elements and inspecting their `style` objects.
- Try enumerating the `style` objects to inspect their properties.
- Then try modifying those properties.
- By this stage, you should now be pretty much an expert at modifying style and appearance.

Support for styles is part of the DOM level 2 specification which the MSIE and Netscape 6.0 browsers support (at least in part). While it's at recommendation stage, it could change. Browsers may implement it incorrectly or introduce bugs inadvertently. Because of that uncertainty, we have covered what seems to work and what we expect to become available, but things may change with new browser versions. There is a lot still to explore and test with Netscape 6.0, and MSIE 6.0 is on the horizon too.

See also:	`CLASS="..."`, `Element` object, `Element.currentStyle`, `Element.runtimeStyle`, `style` object (2), `StyleSheet` object

Element.tagName (Property)

The name of the HTML tag that instantiated this object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myElement*`.tagName`

This property contains a text string with the HTML tag name in upper case for the tag that instantiated the object. This is also some indication of the object class, which will help you to write generic functions that can operate on many kinds of objects in an intelligent way.

The tag name is returned without any left and right carets (greater/less than signs) and in fully DOM compliant implementations will be the same value as the class name of the object. There are a few inconsistencies that prevent this being used reliably as a class name for non DOM specified objects, but within some constraints, it may be useful in that context for determining what kind of object a script is operating on.

It is possible that the `Element` represents a text node. Text nodes are placed between the elements that represent HTML tags. We can then operate on the text in between adjacent HTML tags. With this capability, we probably don't need `innerHTML`, `OuterHTML`, `innerText` and `outerText` properties anymore. However, we will need to get a little bit smarter with our navigation of document trees to locate these text nodes.

If the element does represent a text node, then the `tagName` value is meaningless. Typically a browser will respond with an undefined value which could be difficult to handle. An empty space would have been easier to cope with. You should check the value you get back from this property, unless you are absolutely certain it relates to an HTML tag. If you suspect it might be a text node, then check it before using it or your script will crash.

See also:	`Attribute.nodeName`, `Element` object

Property attributes:

`ReadOnly`.

Element.tagUrn (Property)

The URN value for a tag.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myElement`*`.tagUrn`

This is part of the XMLNS support in MSIE 5.0 and returns a URN value specified in a namespace declaration. This is highly MSIE specific functionality and not standardized for use in other browsers. It is not portable.

Element.title (Property)

The value of the `TITLE` tag attribute for the HTML tag that created this object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myElement`*`.title`
	-	*`myElement`*`.title = `*`aToolTip`
HTML syntax:	`<LINK TITLE="...">`	
Argument list:	*`aToolTip`*	A text string

This is the text that is presented as a 'ToolTip' when the mouse rolls onto an object and pauses there for a few moments. The MSIE browser and the Netscape 6.0 browser will display this text as a small popup comment box. Earlier versions of Netscape prior to version 6.0 did not support this capability, so it should be considered somewhat implementation dependent until the penetration of newer browsers increases.

When aural style sheet support is provided by a browser, this value might be spoken as the mouse rolls over it.

Warnings:

- This property is not the name of the object. The `ID` tag attribute is a more reliable way of making that association, since it assists with the `document.all[]` array construction and indexing on MSIE, and is more portable across browsers when accessing objects associatively.
- Another recommended technique is the DOM compliant `getElementById()` method which in the longer term will become more prevalent.
- This is also not the same as the `altText` on an image, which may also appear as a ToolTip in some browsers.

See also:	`Document.title`, `Element` object, `LINK.title`

Element.uniqueID (Property)

An automatically generated unique ID value for an object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myElement`*`.uniqueID`

You may go to great lengths to map ID values to objects in your documents and still have ID conflicts. This property generates a guaranteed unique ID value which will be static for that object as long as it exists.

However, reloading a page and requesting a unique ID again from objects within it will not yield the same values.

You cannot assign a value to this property.

Property attributes:

`ReadOnly`.

else ... (Keyword)

Part of the `if ... else` conditional code execution mechanism.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0

JavaScript syntax:	-	`if(aCondition){someCode1} else{someCode2}`
Argument list:	*aCondition*	A condition that tests `true` or `false`
	someCode1	Code to be executed if the condition tests `true`
	someCode2	Code to be executed if the condition tests `false`

This is an optional additional statement that can be added to an `if()` condition test to allow for some alternative code to be executed when the condition tests `false`. The `true` case will cause the first block of code to be executed, the `false` case will execute the second (that which follows the `else` keyword).

See also:	`else if( ... ) ...`, Flow control, `if( ... ) ...`, `if( ... ) ... else ...`, Selection statement, `switch( ... ) ... case: ... default: ...`

Cross-references:

ECMA 262 edition 2 – section – 12.5

ECMA 262 edition 3 – section – 12.5

Wrox *Instant JavaScript* – page – 23

else if(...) ... (Idiom)

A technique for stacking `if` conditions in a script.

Although there is no '`else if`' keyword, because the `if` follows the `else` and is separated by a space, the `if` block is treated as if it were a single statement. Here we omit the braces surrounding the statement executed in the `else` condition of a leading `if` block. We then place another in its `else` statement and so on. Like this:

```
if(aCondition){   some code}else if(aCondition){   some code}else if(aCondition){
some code}else{   some code}
```

By putting in the braces, it becomes clearer. Actually, what we have really done is this:

```
if(aCondition){   some code}else{   if(aCondition)   {      some code   }   else
{      if(aCondition)      {         some code      }      else      {
some code      }   }}
```

It is really just a plain old `if() else` block nested several times. By leaving off the braces (against our recommendations otherwise), in this circumstance it actually makes the code clearer.

See also:	Code block delimiter {}, `else ...`, `if( ... ) ...`, `if( ... ) ... else ...`

EM object (Object/HTML)

An object representing the HTML content delimited by the `<EM>` tags.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myEM* = *myDocument*.all.*anElementID*
	IE	*myEM* = *myDocument*.all.tags("EM")[*anIndex*]
	IE	*myEM* = *myDocument*.all[*aName*]
	-	*myEM* = *myDocument*.getElementById(*anElementID*)
	-	*myEM* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myEM* = *myDocument*.getElementsByTagName("EM")[*anIndex*]
HTML syntax:	`<EM> ... </EM>`	
Argument list:	*anElementID*	The ID value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	

`<EM>` tags and the objects that represent them are inline elements. Placing them into a document does not create a line break.

See also:	`Element` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

<EMBED> (HTML Tag)

A mechanism for adding plugin functionality to a web browser.

Inherits from:	Element object

Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the EMBED class. In Netscape they are objects commonly referred to as belonging to the Plugin class.

Because of this, and because each browser supports different properties and methods for them, they will be discussed here as Embed objects for MSIE and Plugin objects for Netscape.

Warnings:

- Note that Netscape can talk to the plugins that are encapsulated with an <EMBED> tag, but the MSIE browser cannot. However MSIE will talks to plugins that are encapsulated with the <OBJECT> tag. Netscape has supported <OBJECT> tags, but somewhat unreliably. Version 6 should be more robust.
- The Macintosh platform does not support the <OBJECT> tag because the MSIE browser really expects and hopes that it will be used to encapsulate an ActiveX object. Since ActiveX objects are written to be compiled into X86 machine code, they can only be used on a Wintel platform.
- Because of this, the MSIE browser on the Macintosh does support some limited communication with <EMBED> tag plugins.
- Trying to find a platform and browser portable compromise for embedding plugins (especially for video playback) is an utterly lost cause right now. You will have to write browser specific modules and use the appropriate one.

See also:	Document.embeds[], Embed object, EmbedArray object, Plugin events, Plugin object

Inheritance chain:

Element object, Node object

Embed object (Object/HTML)

An object that represents an embedded item in MSIE.

Availability:	JavaScript – 1.0 JScript – 3.0 Internet Explorer – 4.0 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myEmbed* = *myDocument*.all.*anElementID*
	IE	*myEmbed* = *myDocument*.all.tags("EMBED")[*anIndex*]
	IE	*myEmbed* = *myDocument*.all[*aName*]
	-	*myEmbed* = *myDocument*.*anElementName*
	-	*myEmbed* = *myDocument*.embeds[*anIndex*]
	-	*myEmbed* = *myDocument*.getElementById(*anElementID*)
	-	*myEmbed* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myEmbed* = *myEmbedArray*[*aName*]
	-	*myEmbed* = *myEmbedArray*[*anIndex*]
	-	*myEmbed* = *myDocument*.getElementsByTagName("EMBED")[*anIndex*]
HTML syntax:	<EMBED>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an `Element` object
Object properties:	accessKey, align, height, hidden, name, palette, pluginspage, readyState, src, tabIndex, units, width	
Event handlers:	onBlur, onClick, onDblClick, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp	

Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the `EMBED` class. In Netscape they are objects commonly referred to as belonging to the `Plugin` class, although they are really implemented as `JavaObject` objects. In MSIE, this is an `ActiveX` object.

There is additional confusion in that there is a `plugins[]` array that belongs to the `document` and another that belongs to the `navigator` object. They both contain collections of objects, but of different types. This is further confused by the fact that the `document.plugins[]` array is another name for the `document.embeds[]` array.

Because of this confusing situation, the best recommendation is that we refer to `document.embeds[]` and `navigator.plugins[]` and quietly ignore the `document.plugins[]` object. Furthermore we shall refer to `Plugin` objects as being something the browser can use to play embedded content and `Embed` objects will be an instance of a plugin that is alive and running in a document.

Warnings:

- Interacting with the properties of `Embed` objects in MSIE seems to work quite reliably. This is not the case with Netscape, which is prone to all kinds of strange behavior. However that may be version and platform specific and could depend greatly on the quality of what you are embedding.
- As an example, Video plugins work really well on one particular platform and are generally less optimal on others. Windows media player is great on Windows and lacking in reliability and quality on other platforms. QuickTime is brilliant on Macintosh and good on Windows, but it still suffers some instability. Real player is pretty good everywhere, but it works better as a player than it does as a plugin. Windows media player works well as a plugin on MSIE. QuickTime is good as both a plugin and a player, but its JavaScripting control is somewhat behind Real player. None of them share the remotest similarity as far as properties, parameters or JavaScript API calls are concerned. If only we could have some compatible video players, we could solve a lot of the embedding problems.

See also:	`<EMBED>`, `Document.embeds[]`, `Element` object, `EmbedArray` object, `Input.accessKey`, `Plugin` object

Property	JavaScript	JScript	N	IE	Opera	NES	DOM	HTML	Notes
`accessKey`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	2.0 +	-	-	Warning
`align`	-	3.0 +	-	4.0 +	-	-	-	-	-
`height`	-	3.0 +	-	4.0 +	-	-	-	-	-
`hidden`	-	3.0 +	-	4.0 +	-	-	-	-	-
`name`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	-	-
`palette`	-	3.0 +	-	4.0 +	-	-	-	-	ReadOnly
`pluginspage`	-	3.0 +	-	4.0 +	-	-	-	-	ReadOnly
`readyState`	-	3.0 +	-	4.0 +	-	-	-	-	ReadOnly
`src`	-	3.0 +	-	4.0 +	-	-	-	-	-
`tabIndex`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	2.0 +	-	-	Warning
`units`	-	3.0 +	-	4.0 +	-	-	-	-	-
`width`	-	3.0 +	-	4.0 +	-	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	NES	DOM	HTML	Notes
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	-	Warning
`onClick`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	NES	DOM	HTML	Notes
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
onMouseOver	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

Embed.align (Property)

Controls the alignment of an embedded plugin relative to its surrounding objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEmbed*.align

The alignment of the `plugin` object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

Embed.height (Property)

The height of a plugin's extent rectangle.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEmbed*`.height`

The space reserved for the plugin is defined by an extent rectangle that surrounds the space occupied by it even before it is loaded. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

Embed.hidden (Property)

A flag indicating whether a plugin is hidden or not.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myEmbed*`.hidden`

Setting this property to `true` will hide the plugin, and a `false` value will reveal it.

Embed.name (Property)

This corresponds to the `NAME` attribute of the `<EMBED>` tag.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myEmbed*`.name`

Objects are identified either by the `NAME` HTML tag attribute or by the `ID` HTML tag attribute.

Netscape 4 shows a marginal preference for the `name` property while MSIE seems slightly better disposed towards the `ID` property. However, in many cases, both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace. Netscape 6.0 is DOM based and works well with `ID` values.

Embed.palette (Property)

The palette for use with a plugin.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEmbed*.palette

In the Windows environment, there are two alternative palettes that you can apply to an embedded object. This property reflects the value of the `PALETTE` HTML tag attribute on the `<EMBED>` tag that instantiates this object.

It should contain one of these values to indicate which palette is being used:

- foreground
- background

Property attributes:

`ReadOnly`.

Embed.pluginspage (Property)

The URL of a page where the plugin can be obtained if it is not yet installed in the browser.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEmbed*.pluginspage

This contains a URL value where the plugin can be obtained if you do not have it installed in your browser.

Property attributes:

`ReadOnly`.

Embed.readyState (Property)

The downloading status disposition of the plugin associated with the `<EMBED>` tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEmbed*.readyState

This property reflects the loading status of an `EMBED` object.

Sometimes, you can design scripts to execute while the document is downloading. Inline scripts for example. At that time, you may even be able to trigger interval timed deferred executions as well.

If it is important to know when the document has completed loading, you can check this property for one of the following values:

State:	Value:
uninitialized	The object is first instantiated but has not begun loading.
loading	The object has commenced loading.
loaded	The object has completed loading.
interactive	The object is loaded but not yet closed, and is ready to handle interaction.
complete	The object body has been closed and the loading is finished.

An object may not need to reflect the complete status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an `OBJECT` object that represents an `<OBJECT>` tag until the `<BODY>` has completed loading. This is because the `ActiveX` object construction requires a complete document body structure to attach itself to.

Every time this `readyState` value changes, it triggers an `onReadyStateChange` event call-back.

See also:	`onReadyStateChange`

Property attributes:

`ReadOnly`.

Embed.src (Property)

The URL of an external file that the plugin may use.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myEmbed`*`.src`

Some plugins may need to access a supplementary data file from the server. It is good practice to abstract such data values from the code itself and so a means of passing this parameter value in from outside is necessary. The `SRC` HTML tag attribute is reflected into this property and is provided as a somewhat standardized means of passing one of the parameter values most likely to be defined.

Embed.units (Property)

The unit of measure for the height and width of the plugin extent rectangle.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEmbed*`.units`

This value appears to be derived from the measurement units used in style sheets.

However, it does not appear to work at all and has no observable effect on the browsers. There are several possible values for this, but they are not supported uniformly across all browsers and platforms. This may be better supported in the future.

The best solution for now is to omit this attribute and always measure `<EMBED>` tag objects in pixels.

The following values are allegedly available:

- pixels
- en
- em

Embed.width (Property)

The width of the plugin extent rectangle.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEmbed*`.width`

The space reserved for the plugin is defined by an extent rectangle that surrounds the space occupied by it even before it is loaded. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

EmbedArray object (Object/browser)

A more appropriate name for a `PluginArray` that contains a collection of plugins within the current document.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0

Inherits from:	`Object` object
JavaScript syntax:	- *myEmbedArray* = *myDocument*.embeds
	- *myEmbedArray* = myDocument.*plugins*
Object properties:	`length`

A collection of objects, each one representing a plugin that is embedded in the current page. The object being referred to is not the embedded data, but the plugin module that plays it.

Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the `EMBED` class. In Netscape they are objects commonly referred to as belonging to the `Plugin` class although they are really implemented as `JavaObject` objects. In MSIE, this is an `ActiveX` object.

There is additional confusion in that there is a `plugins[]` array that belongs to the `document` and another that belongs to the `navigator` object. They both contain collections of objects but of different types. This is further confused by the fact that the `document.plugins[]` array is another name for the `document.embeds[]` array.

Because of this confusing situation the best recommendation is that we refer to `document.embeds[]` and `navigator.plugins[]` and quietly ignore the `document.plugins[]` object. Furthermore we shall refer to `Plugin` objects as being something the browser can use to play embedded content and `Embed` objects will be an instance of a plugin that is alive and running in a document.

Warnings:

- Beware of confusion between `document.plugins` and `navigator.plugins`. The former relates to the plugins currently used in the document while the latter lists the plugins currently available and supported by the browser.

See also:	`<EMBED>`, `Collection` object, `Document.embeds[]`, `Document.plugins[]`, `Embed` object, `JavaObject` object, `Plugin` object, `PluginArray` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`length`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	ReadOnly

Inheritance chain:

`Object` object

EmbedArray.length (Property)

The number of `plugin` objects in the document.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myDocument*`.embeds.length` - *myDocument*`.plugins.length`
See also:	`EmbedArray` object, `Collection.length`

Property attributes:

`ReadOnly`.

Embedded JavaScript (Definition)

JavaScript available inside your application.

Embedded interpreters are available to buy or download as open source projects.

With an embedded interpreter, you can break your application into components and integrate them together with fragments of JavaScript. This allows you to flexibly reconfigure and extend an application very easily.

See also:	Host environment, Platform

Cross-references:

Wrox *Instant JavaScript* – page – 5

Emboss() (Filter/visual)

Displays the image content of the HTML element as if it were an embossed effect.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

filter – `Emboss()`

Empty statement (;) (Statement)

A no-op 'do-nothing' statement.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0
JavaScript syntax:	- ;

An empty statement is signified by a semi-colon on a line by itself or a semi-colon following a semi-colon with no executable statements in between them.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

Warnings:

- If you intentionally place an empty statement into your script, you should put a comment adjacent to it to make sure people realize you did it on purpose. They may remove it otherwise and you may have had a functional reason for putting it there such as to aid the parsing of the text inside a `SCRIPT` object.
- This sails awfully close to the territory of self-modifying code which programmers have always loved to do and preachers of good programming style have said is a very bad thing.
- On the other hand, JavaScript supports the `eval()` function and you can't get closer to self modifying code than that.

See also:	Associativity, Operator Precedence, Semi-colon (;), Statement

Cross-references:

ECMA 262 edition 2 – section – 12.3

ECMA 262 edition 3 – section – 12.3

enableExternalCapture() (Method)

Part of the Netscape 4 event propagation complex.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`enableExternalCapture()`
	-	*`myWindow`*`.enableExternalCapture()`

Refer to:

`Window.enableExternalCapture()`

encodeURI() (Function)

This ECMA defined function can be used to encode an entire URI value that can then be decoded with the `decodeURI()` function.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`encodeURI(`*`aURI`*`)`
Argument list:	*`aURI`*	An unencoded URI

The `encodeURI()` function computes a new version of the string value it is passed. The new version has certain characters replaced with hexadecimal escape sequences. The string is expected to conform to the UTF-8 profile.

All character codes other than letters, numbers or a small set of special characters will be escaped.

These special characters are not transformed:

- Minus
- Underscore
- Period
- Exclamation-mark
- Tilde
- Single quote
- Opening and closing parentheses

Note that the hash character is also not encoded by this function.

See also:	`decodeURI()`, `decodeURIComponent()`, `encodeURIComponent()`, `escape()`, `unescape()`, URI handling functions

Cross-references:

ECMA 262 edition 3 – section – 15.1.3.3

encodeURIComponent() (Function)

This ECMA defined function can be used to encode a URI component value that can then be decoded with the `decodeURIComponent()` function.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- `encodeURIComponent(`*`aComponent`*`)`
Argument list:	*`aComponent`* A URI component to be encoded

The `encodeURIComponent()` function is used to encode individual components belonging to a URI. You will need to deconstruct the URI yourself if you intend to call this function from your own scripts.

See also:	`decodeURI()`, `decodeURIComponent()`, `encodeURI()`, `escape()`, `unescape()`, URI handling functions

Cross-references:

ECMA 262 edition 3 – section – 15.1.3.4

Engrave() (Filter/visual)

An effect that is the opposite of the embossed image appearance.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

filter – `Engrave()`

Enquiry functions (Definition)

A means of determining value types.

The following enquiry functions are built in:

- `isFinite()`
- `isNaN()`

This operator can behave as if it were a function:

- typeof

These enquiry functions are available in other languages and we have documented them elsewhere with example scripts to simulate their functionality:

- `isalnum()`
- `isalpha()`
- `iscntrl()`
- `isdigit()`
- `isgraph()`
- `islower()`
- `isodigit()`
- `isprint()`
- `ispunct()`
- `isspace()`
- `isupper()`
- `isxdigit()`

See also:	`isAlnum(),isAlpha(),isCtrl(),isDigit(),isFinite(), isGraph(),isLower(),isNaN(),isODigit(),isPrint(), isPunct(),isSpace(),isUpper(),isXDigit(),typeof`

Entity object (Object/DOM)

An entity is an item in an XML document. This object encapsulates the XML entity.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Inherits from:	`Node` object

JavaScript syntax:	-	*myEntity* = new Entity()
Object properties:	notationName, publicId, systemId	

DOM level 3 is expecting to add the following properties:

- actualEncoding
- encoding
- version

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
notationName	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly.
publicId	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly.
systemId	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly.

Inheritance chain:

Node object

Entity.notationName (Property)

If an entity is unparsed, then this will be the name of the notation for that entity, otherwise it is null.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myEntity*.notationName

Property attributes:

ReadOnly.

Entity.publicId (Property)

The public identifier associated with the entity if it was specified when the entity was instantiated.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myEntity*.publicId

Property attributes:

ReadOnly.

Entity.systemId (Property)

The system identifier associated with the entity if it was specified when the entity was instantiated.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myEntity*.systemId

Property attributes:

ReadOnly.

EntityReference object (Object/DOM)

A reference to an entity object in an XML document.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Inherits from:	Node object	
JavaScript syntax:	N	*myEntityReference* = *myDocument*.createEntityReference(*aName*)
Argument list:	*aName*	The name of the entity to reference

See also:	Document.createEntityReference()

Inheritance chain:

Node object

enum (Reserved word)

Reserved for future language enhancements.

The provision of this keyword suggests that future versions of ECMAScript may support enumerated data types.

See also:	Reserved word, Type

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Enumeration constant (Definition)

Possible future functionality to provide enumerated data types.

Refer to:

`enum`

Enumerator object (Object/JScript)

A special object supported by MSIE for processing collections of objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myEnumerator* = Enumerator
	IE	*myEnumerator* = new Enumerator(*aCollection*)
Argument list:	*aCollection*	The collection to be enumerated
Object properties:	constructor	
Object methods:	atEnd(),item(),moveFirst(),moveNext()	

An `Enumerator` object provides a way to enumerate through all the objects in a collection (aka Array). You can create a new `Enumerator`, giving it your collection as an argument and can then access the items in the collection in a more sophisticated way than simply using a `for` loop.

You can use the `enumerator` to cycle through the items in a collection in much the same way as a `for( ... in ... )` loop would enumerate the properties. However, in some collections, objects have more than one entry. They may have an indexed entry and an associative entry. An `enumerator` should traverse the collection visiting each item only once. A `for` loop may visit objects several times.

The available set of methods and properties is somewhat limited compared with `enumerator` objects in other languages.

Because this is only available on MSIE and is severely dysfunctional on the Macintosh version of MSIE 5, its use is somewhat limited from the portability point of view. It is recommended that you avoid using it for the time being. Later, when it is more widely and reliably available, it may be more useful.

Do not confuse DOM `NodeList` arrays with `Enumerator` or `Collection` objects. The `NodeListitem()` method is subtly different to the `Enumerator.Item()` method.

Warnings:

- When tested on MSIE 5 for Macintosh, this object exhibited some very odd behavior.
- When passed a `FormArray`, it complained that it was not a `Collection` object. When passed a `Collection` object (`document.all`) it still complained. However, when passed an `Array` object, it was happy to accept it. It would allow a new `Enumerator` to be created with no argument being passed to its constructor function.
- When examined, its constructor reported that it was a reference to a `Date` object.
- The object may only be usable on MSIE on the Windows platform until a later version of MSIE supports a corrected implementation.
- The naming convention for methods and properties of this object are capitalized in a very untypical way and you need to be aware of this in case you have trouble getting your `enumerator` to work properly.

Example code:

```
// Instantiate a file system object
myFileSystem = new ActiveXObject("Scripting.FileSystemObject");

// Create an enumerator
myEnum = new Enumerator(myFileSystem.Drives);

// Traverse the Drives collection via the enumerator
for(; !myEnum.atEnd(); myEnum.moveNext())
{
   processDrive(myEnum.item());
}

// A function to do something with each disk drive
function processDrive(aDrive){...}
```

See also:	`Collection` object, `Files` object, `NodeList` object

Property	JavaScript	JScript	N	IE	Opera	Notes
`constructor`	-	3.0 +	-	4.0 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	Notes
`atEnd()`	-	3.0 +	-	4.0 +	-	-
`item()`	-	3.0 +	-	4.0 +	-	-
`moveFirst()`	-	3.0 +	-	4.0 +	-	-
`moveNext()`	-	3.0 +	-	4.0 +	-	-

Enumerator() (Constructor)

A constructor function for creating new `Enumerator` objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Enumerator` object	
JavaScript syntax:	IE	`new Enumerator()`
	IE	`new Enumerator(`*`aCollection`*`)`
Argument list:	*`aCollection`*	A collection of objects to be enumerated

This is the constructor function for creating new `enumerator` objects. Use it with the `new` operator to manufacture an `Enumerator` and then store the reference to it in a variable.

Enumerator.atEnd() (Method)

A method that returns a flag indicating the end of the collection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myEnumerator`*`.AtEnd()`

In your enumeration loop, you can test this method and exit the loop if it returns the Boolean `true` value.

Enumerator.constructor (Property)

A reference to the `constructor` object for the `Enumerator`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Enumerator` object	
JavaScript syntax:	IE	*`myEnumerator`*`.constructor`

You can access the constructor for an existing `Enumerator` object here.

You can use this as one way of creating `Enumerator` objects, although it is more popular to use the `new Enumerator()` technique.

This property is useful if you have an object that you want to clone, but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

Warnings:

- On the Macintosh version of MSIE 5, this property yields a reference to the `Date()` constructor object. This is obviously a bug and renders the `Enumerator` unusable on this platform.

Enumerator.item() (Method)

A reference to the current item in the collection. This method returns the object from the collection that the `enumerator` is currently accessing.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`User defined`	
JavaScript syntax:	IE	*myEnumerator*.item(*anIndex*)
	IE	*myEnumerator*.item(*aSelector*)
	IE	*myEnumerator*.item(*aSelector*, *anIndex*)
Argument list:	*anIndex*	A zero based index into the collection
	aSelector	A textual value that selects all matching objects

Refer to:

`Collection.Item()`

Enumerator.moveFirst() (Method)

Resets the enumerator to point at the first item in the collection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myEnumerator*.MoveFirst()

This relocates the enumerator so that it accesses the first item in the collection.

Enumerator.moveNext() (Method)

Moves the enumerator to the next item in the collection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myEnumerator*.MoveNext()

This indexes the `enumerator` onwards to the next item in the collection that has not yet been visited.

Environment (Definition)

The environment is the computing context in which the script is executed.

There are a variety of different environments in which a script may be executed. At the time of writing, a JavaScript script could be operating in any of these distinctly different environments:

- A Netscape web browser
- A MSIE browser
- Several other new web browsers
- A server CGI environment
- A desktop application environment
- A UNIX shell
- A WebTV set top box
- A Liberate TV Navigator set top box
- A WAP/WScript mobile phone
- An Adobe PDF file reader
- An embedded web browser built-into consumer products

Each of these has certain advantages and constraints. Most offer special facilities native and unique to that hosting environment.

In general, you should be able to determine which of these environments you are operating in. However, there is no standardized way to detect this at present.

There may be range limits on values in certain environments and certainly there will be 'bugs' in the implementations that are platform specific. It is also very likely that functionality will be more or less incomplete in some environments – mostly depending on the maturity of the implementation.

See also:	Character display semantics, Character set, Execution environment, Host environment, Limits, Script termination

Cross-references:

Wrox *Instant JavaScript* – page – 5

Equal to (==) (Operator/equality)

Compares two operands for equality.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`anOperand1 == anOperand2`*
Argument list:	*`anOperand1`*	A value that can reasonably be compared
	`anOperand2`	A value that can reasonably be compared with *`anOperand1`*

The result is the Boolean value `true` if `anOperand1` is numerically or lexically equal to `anOperand2`, otherwise `false` is returned.

The equality operator is not always transitive. For example, two string objects may represent the same string value. Comparing the objects for equality will yield `false` because the references to the objects are being compared and not the object values themselves. However, forcing a string comparison may in fact yield a `true` value when testing for equality. Do this by converting the objects as part of the comparison process by type conversion or `valueOf()` methods.

Numeric values may require rounding to take place and testing for equality may be accurate down to a precision finer than your script cares about. You can set a precision value in a variable, then subtract the value you are comparing with from the absolute value of the comparee. If the difference is smaller than your precision value, then the two values are close enough to yield a match.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

Refer to the Equality expression topic for a discussion on the ECMA standard definition of the equality testing rules.

Warnings:

- Be careful not to confuse the single equals with the double equals. Placing a single equals in place of a test for equality will assign the right hand value to the left hand operand and clobber the value accidentally. It will also force the relational expression to yield `true` as well. The interpreter may be forgiving enough that a run-time error isn't generated, but the side effects could be subtle and make it hard to diagnose the cause.
- A triple equals sign further complicates things as it is a test for identical type as well as equal value.

Example code:

```
// Fuzzy matching of numeric values
function almost_equal(aValue1, aValue2)
{
   var myPrecision = 1e-10;
   if((Math.abs(aValue1 - aValue2)) < myPrecision)
   {
      return(true);
   }
   return(false);
}
```

See also:	ASCII, Type conversion, `typeof`, Unicode

Cross-references:

ECMA 262 edition 2 – section – 11.9.1

ECMA 262 edition 3 – section – 11.9.1

Wrox *Instant JavaScript* – page – 36

Equality expression (Definition)

An expression that tests for equality or not.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

Equality expressions are a special case of Relational expressions. They deal strictly with equality or non-equality.

There are two equality operators that you can use to make an equality expression:

- The == operator tests for equality.
- The != operator tests for inequality.

As a general rule, equality expressions will yield a `true` or `false` result in a more forgiving way than relational expressions. Passing `NaN`, undefined and `null` values to relational expressions may yield undefined values as results, where an equality expression would still return a Boolean value.

The comparisons between objects are likely to be a shallow comparison. If you are comparing two objects of the same type, the comparison logic will check to see if you are referring to the same instance. That is a test for identity and not equality. A deeper comparison might compare two similar objects on a property by property basis. They wouldn't be identical, but they may be equivalent. You could simulate this with a script function that returns `true` or `false` having done a deep comparison.

Tests for equality require further deductive reasoning on the part of the interpreter. The values are converted to their preferred types. If the types are the same, then the values can be compared easily either as Numbers or Strings. If the types are different, the further conversion is necessary before the comparison can be completed. In that case, Boolean become Numbers as do any other non-numeric values and numeric comparison predominates.

Comparing `null` with undefined values does not require any conversion and they will compare equal.

Comparing values with `null` can expose some bugs in earlier implementations. It may be safer to rely on Boolean conversions and simply test for `true` or `false`.

The ECMA standard (edition 3) sets out the rules for testing two values for equality (somewhat simplified):

- If they are of different types, they are unequal – return `false`.
- If the type of the first argument is undefined or `null` return `true`.
- For numeric values `NaN` in either case return `false`.
- Positive and negative zeros are represented differently internally but are equal.
- Otherwise equal numbers return `true` and unequal numbers return `false`.
- For strings, the two values must contain the same sequence of characters for them to be equal.
- Boolean values must be equal and they have no special states to consider.
- Object references must refer to the same object instance.
- The values `null` and undefined are considered to be equal.
- If necessary, `ToNumber()` and `ToPrimitive()` functions are called to coerce objects when one value is a primitive and the other is an object.
- If all of the above fail to match equal then a false value is returned.

It seems odd to assume equality to be true if the type of the arguments is not absolutely clear, but this may be necessary to allow implementations some flexibility in the internal representation of values which may not have a defined type.

It might also be strange to see that if either value is `NaN` then a `false` value is returned. This is necessary because `NaN` is indeterminate. It is not a specific value and can be caused by a variety of circumstances when an expression yields an out of bounds value. There is no guarantee that two individual `NaN` values resulted from the same circumstance and hence they are assumed to be unequal.

Referring to the same object also includes joined objects which are an internal mechanism for sharing functionality between objects. This is not exposed to the script interface.

Warnings:

- Earlier versions of MSIE and Netscape exhibited bugs in the comparison logic between the results of comparisons involving `NaN`, `null` and `0`, where type conversions led to inconsistent behavior.
- Since historically there are known bugs in comparisons involving `null` values, you should avoid using them. The following:

```
if(a){   someCode}
```

may be more reliable in some implementations than:

```
if(a != null){   someCode}
```

Example code:

```
// Force a string comparison
myResult = (a+'' == b+'');
alert(myResult);


// Force a numeric comparison
myResult = (a-0 == b-0);
alert(myResult);

// Force a boolean comparison
myResult = (!a == !b);
```

See also:	Equal to (==), Equality operator, Expression, Identically equal to (===), NOT Equal to (!=), NOT Identically equal to (!==), Relational expression, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 11.9

ECMA 262 edition 3 – section – 11.9

Wrox *Instant JavaScript* – page – 39

Equality operator (Definition)

An operator that tests for equality or not.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

There are two equality operators:

- The == operator tests for equality.
- The `!=` operator tests for inequality.

Equality operators deal exclusively with the test for the operands being equal to one another. They yield a `true` or `false` result and are generally considered as part of the relational operator set since they are most often used in the same circumstances.

Testing two operands for equality follows these basic rules:

If the native types of the two operands are not the same, then the values are not equal unless a numeric coercion yields equal values from both sides or the operands both yield `null`/undefined values.

If the type of the left operand is undefined or `null`, it is assumed to be equal to the right operand.

If either of the operands is `NaN`,then they are assumed to be not equal.

Positive and negative zero are equal values.

Boolean values must be identical values to be considered equal.

Two strings of the same length containing the same character sequence (identical copies of one another in terms of Unicode character code points) are assumed to be equal.

References to the same object test as equal. References to two objects containing the same property values, even though they may be copies of one another, are not equal.

Comparing different types of operands can use coercion techniques to force the comparison to be conducted according to string, numeric or Boolean rules.

If one of the operands is an object and the other is not, then the object is converted to a primitive.

String comparisons can be by concatenating values of other types to an empty string. It may help to use the parentheses to guarantee that precedence is established as you intend it to be. Here is a forced string comparison:

```
("" + a) == ("" + b)
```

Numeric comparisons can be forced by subtracting zero. Again, grouping operators help to establish the desired precedence:

```
(a - 0) == (b - 0)
```

Boolean comparisons can be forced by performing a logical NOT on both operands, in which case precedence control with grouping operators may not be as necessary as it is with the addition and concatenation operators:

```
!a == !b
```

The values `null` and `undefined` are generally considered to be equal although they are distinctly different values. This is because early implementations did not support an explicit `undefined` value and allowed for it to have the same meaning as the `null` value.

Comparing strings is done simply according to the Unicode character code point values and does not take into account any of the more subtle semantic meanings of those characters as defined in the Unicode version 2.0 specification.

Refer to the identity operators for a more exact comparison taking data type into account.

Warnings:

- Be careful not to miss one of the equals signs when testing for equality. You can accidentally assign to an LValue and not realize it.

See also:	= (Assign), Associativity, Equal to (==), Equality expression, Identically equal to (===), NOT Equal to (!=), NOT Identically equal to (!==), Operator, Operator Precedence, Relational operator

Cross-references:

ECMA 262 edition 2 – section – 11.9

ECMA 262 edition 3 – section – 11.9

Error (Definition)

That which happens when your script fails to execute properly.

Availability:	ECMAScript edition – 2

The ECMA standard dictates that a compliant implementation will detect and alert on all errors in the code even if the error is in a section of code that is not executed. Such dead code might be in a conditional block for which the condition could never be true. Nevertheless, a compile time warning should be generated.

This means that you should not rely on placing code within an `if(false)` block to prevent its execution. Instead, you should comment out the code to achieve the same effect.

This behavior is generally implementation dependent due to the different ways that compilers and interpreters may be designed and built. It also changes as browser versions evolve and of course the kind of error that is thrown can also affect when and how it is detected and managed.

See also:	Diagnostic message, Error handler, Mathematics

Cross-references:

ECMA 262 edition 2 – section – 16

ECMA 262 edition 3 – section – 16

Error events (Definition)

A class of events that are triggered by errors.

HTML syntax:	`<IMG onError="...">`

There is an `onError` event defined as part of the set of events supported by JavaScript. However, as of JavaScript version 1.2, the `onError` event is actually only supported by `Image` objects when defined in HTML tag attributes. You can add an error handler to a `window` object with the `onerror` property.

The `<IMG onError="...">` tag element allows an error during image loading to be handled gracefully.

Attaching an error handler to a `window` with the `window.onerror` property allows JavaScript errors to be intercepted.

Error events have a slightly different parameter passing API so although they are events, they are slightly different to the rest of the event model.

See also:	Error handler, Event handler, Event model, `onError`, `Window.onerror`

Error handler (Interface)

Triggered by an error in the JavaScript execution.

JavaScript syntax:	- `function` *`anErrorHandler`*`(`*`aMessage`*`, `*`aURL`*`, `*`aLine`*`) { `*`someCode`*` };`	
Argument list:	*`aLine`*	The line number where the error occurred
	`aMessage`	The text of the error message
	`anErrorHandler`	An identifier name for the function
	`aURL`	The URL of the document containing the error
	`someCode`	The script source for the error handler

This is a means of trapping the errors in any script. You can activate an error handler in various ways. Placing this line at the top of your script will trap all errors within that script:

```
self.onerror = function() { return true };
```

Because the script always returns true, no error dialog will ever be displayed.

Here is a pseudo code example of a better technique.

```
function HandleErrors(aMessage, aURL, aLine){   //If we can handle the error with
a piece of code      return true;   //Else we can't do anything      return
false;}self.onerror = HandleErrors;
```

An error handler should return true if the host environment can safely ignore the error and `false` if the environment needs to handle the error as normal.

See also:	Error, Error events, Semantic event, `Window.onerror`

Error handling (Definition)

A mechanism for dealing with errors.

Error handling in compiled languages happens in two stages.

First, compile time messages tell you about syntax errors. Secondly, when you run the application, you get run-time errors. These are generally semantic problems.

JavaScript is an interpreted language and as such compilation and run-time happen one after the other when the script is executed. Some implementations parse and tokenize the entire script before executing any of it, others may parse and execute a line at a time.

If you are handling errors in server-side code, you may be able to make use of the following methods/properties of the Netscape Enterprise Server's server-side objects. These will tell you about errors that have just occurred:

- `majorErrorCode()`
- `majorErrorMessage()`
- `minorErrorCode()`
- `minorErrorMessage()`

The status codes returned by NES database calls imply that you should inspect these error code values when the status code is 5 or 7 (depending on the database type).

Warnings:

- In an effort to make the Internet more accessible to inexperienced users, you may find some browsers, especially those in TV set-top boxes, handle errors silently. A fatal error in a script may halt the script and indeed stop any others executing on the page. However, the user will not receive any warning that this has happened. This is certainly the case with the WebTV set-top box and may be true of others.

See also:	`Connection.majorErrorCode()`, `Connection.majorErrorMessage()`, `Connection.minorErrorCode()`, `Connection.minorErrorMessage()`, Constraint, `database.majorErrorCode()`, `database.majorErrorMessage()`, `database.minorErrorCode()`, `database.minorErrorMessage()`, `DbPool.majorErrorCode()`, `DbPool.majorErrorMessage()`, `DbPool.minorErrorCode()`, `DbPool.minorErrorMessage()`, Diagnostic message, Input-output, JellyScript, Script termination, Status code

Error object (Object/core)

An object that represents a custom error condition.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

JavaScript syntax:	-	*myError* = new Error()
	-	*myError* = new Error(*aNumber*)
	-	*myError* = new Error(*aNumber*, *aText*)
Argument list:	*aNumber*	An error number
	aText	A text describing the error
Object properties:	constructor, description, message, name, number, prototype	
Object methods:	toString()	

This object is provided to create custom error codes for your application. The ECMA standard (edition 3) describes them as objects that are thrown as exceptions when a run-time error occurs.

These objects are passed as the first argument of the `catch()` function in a `try ... catch` structure where you can inspect them and deal with the error.

Example code:

```
// Force an error condition
myError = new Error(100, "My user defined error text")
try
{
   throw myError;
}
catch(anErr)
{
   confirm(anErr.description);
}
finally
{
   alert("Sorted");
}
```

See also:	`catch( ... )`, `EvalError` object, `RangeError` object, `ReferenceError` object, `SyntaxError` object, `throw`, `try ... catch ... finally`, `TypeError` object, `URIError` object

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
constructor	1.5 +	5.0 +	6.0 +	5.0 +	-	-	3 +	DontEnum.
description	-	5.0 +	-	5.0 +	-	-	-	Warning
message	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	Warning
name	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
number	1.5 +	5.0 +	6.0 +	5.0 +	-	-	-	-
prototype	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	2.0 +	3 +	ReadOnly, DontDelete, DontEnum.

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`toString()`	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	2.0 +	3 +	-

Cross-references:

ECMA 262 edition 3 – section – 15.1.4.9

ECMA 262 edition 3 – section – 15.11.1

ECMA 262 edition 3 – section – 15.11.2

Error() (Constructor)

An `Error` object constructor.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	`new Error()`
	-	`new Error(`*`aNumber`*`)`
	-	`new Error(`*`aNumber`*`, `*`aText`*`)`
Argument list:	*`aNumber`*	An error number
	`aText`	A text describing the error

The `Error()` constructor can be called with the `new` operator or as a function.

The initial value of `Error.prototype.constructor` is the built-in `Error()` constructor.

The two arguments supplied to the `Error()` constructor describe the error number and a textual description. They are all optional.

A `prototype` property is automatically created in case the `Error` object is used as a constructor at some future time.

See also:	Constructor function, `constructor` property, `Error.prototype`

Cross-references:

ECMA 262 edition 3 – section – 15.11.2

Error() (Function)

An `Error` object constructor.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	`Error()`
	-	`Error(`*`aNumber`*`)`
	-	`Error(`*`aNumber`*`, `*`aText`*`)`
Argument list:	*`aNumber`*	An error number
	`aText`	A text describing the error

When the `Error()` constructor is called as a function, it creates and initializes a new `Error` object. The function call `Error()` is equivalent to the expression `new Error()` with the same arguments.

See also:	`Error.prototype`

Cross-references:

ECMA 262 edition 3 – section – 15.11.1

Error.constructor (Property)

A reference to a constructor object.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Error` object	
JavaScript syntax:	-	*`myError`*`.constructor`

The initial value of `Error.prototype.constructor` is the built-in `Error()` constructor.

You can use this as one way of creating `Error` objects, although it is more appropriate to use the `new Error()` technique.

This property is useful if you have an object that you want to clone, but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	`Error.prototype`

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 3 – section – 15.11.4.1

Error.description (Property)

A property that corresponds to the description argument in the constructor function.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myError`*`.description`

A human readable textual description of the custom error is available in this property. You can assign a new value if you wish. Reading the property yields the value that was passed in the description argument to the constructor function unless it has been changed since then.

Warnings:

- The ECMA standard describes this property as `Error.message`. MSIE supports it as the `description` property.

Error.message (Property)

A property that corresponds to the message argument in the constructor function.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myError`*`.message`

A human readable textual description of the custom error is available in this property. You can assign a new value if you wish. Reading the property yields the value that was passed in the description argument to the constructor function unless it has been changed since then.

Warnings:

- The ECMA standard describes this property as `Error.message`. Netscape 6.0 claims to be completely compliant and therefore should support this property.
- MSIE supports the `description` property which is functionally identical.

Cross-references:

ECMA 262 edition 3 – section – 15.11.4.3

Error.name (Property)

The name of an `Error` object can be accessed with this property.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	- *myError*.name	
	- *myError*.name = *aString*	
Argument list:	*aString*	The new name value for the `Error` object

The initial value for this property is the string primitive "`Error`". The ECMA standard suggests that this property can be assigned with a different name for each instance of the `Error` object.

For native errors that are subclassed from this `Error` object, the `name` property will be the name of that object class and will be one of the following:

- `EvalError`
- `RangeError`
- `ReferenceError`
- `SyntaxError`
- `TypeError`
- `URIError`

Cross-references:

ECMA 262 edition 3 – section – 15.11.4.2

Error.number (Property)

A property that corresponds to the number argument in the constructor function.

Availability:	JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myError`*`.number`

This is the error number your custom error will be registered under. Unless you change it by assigning a new value to this property, it will contain the value that was passed in the number argument to the constructor function.

The number is a 32 bit value consisting of a facility code and an error number. The upper 16 bits is the facility code and the lower 16 bits the error number within that facility.

Error.prototype (Property)

The prototype for the `Error` object that can be used to extend the interface for all `Error` objects.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Error` object	
JavaScript syntax:	-	`Error.prototype`
	-	*`myError`*`.constructor.prototype`

This is the `Error` prototype object belonging to the `global` object and from which all `Error` objects are descended.

The initial value of `Error.prototype` is the built-in `Error` prototype object.

The value of the `prototype` property of an `Error` object is used to initialize child objects when that `Error` object is used as a constructor.

The following properties are inherited from the `Error.prototype`:

- `Error.prototype`
- `Error.constructor`
- `Error.name`

- `Error.message` (according to ECMA)
- `Error.description` (according to Microsoft)

The following methods are inherited from the `Error.prototype`:

- `Error.toString()`

The example demonstrates how to provide extensions to all instances of this class by adding a function to the prototype object to emulate the ECMA standard message property that is lacking in some implementations.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that provides ECMA compliance
function message()
{
   return this.description;
}

// Register the new function
Function.prototype.message = message;

// Create an Error object and test the new property accessor
var myError = new Error(100, "My error message");
document.write(myError.message);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Error()`, `Error()`, `Error.constructor`, `Error.toString()`, `prototype` property

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 3 – section – 15.11.3.1

Error.toString() (Method)

Returns a string primitive version of an `Error` object.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myBoolean*.toString()

The value of the object is converted to a string value that represents its value. The value returned is implementation dependent.

See also:	`Cast` operator, `Error.prototype`, `toString()`

Cross-references:

ECMA 262 edition 3 – section – 15.11.4.4

Escape sequence (\) (Definition)

A means of defining characters that cannot easily be typed on a keyboard.

Availability:	ECMAScript edition – 2 Opera – 3.0

The string delimiter characters present problems if you need to include them inside a string. Typically you may want to include a single quote as an apostrophe, as in the contraction of do not to don't. Or you may want to enclose a spoken comment inside double quotation marks.

Because the single and double quotes are generally interchangeable in JavaScript, in most cases the problem is easy to solve.

To use " inside a string, you can do it like this:

```
myString = 'A man said "hello" to me';
```

Using a double quote to include single quotes in your string, you might do this:

```
myString = "Don't do that";
```

Using the backslash escape delimiter we can safely use any combination of quotes inside either sort. Like this:

```
myString = "A man said \"hello\" to me";

myString = 'Don\'t do that';
```

A further use for this is to escape a single dot character in regular expressions. This allows you to search for full stops in the source text because a dot in a Regular Expression is a wildcard character match. This technique is useful for other meta symbols in the regular expression set. Refer to the regular expression topic for more details.

Just to illustrate here is a simple example. This regular expression matches any single character at the start of a line:

```
/^./
```

This variant matches only full stops or periods at the start of the line:

```
/^\./
```

An escape sequence is a series of characters that taken together describe a character that cannot normally be represented in the code set that the source is written in, or may be difficult to type on keyboards that are not specially designed for international character sets.

Escape Sequence	Name	Symbol
\"	Double Quote	"
\'	Single Quote (Apostrophe)	'
\\	Backslash	\
\a	Audible alert (MSIE displays the letter a)	<BEL>
\b	Backspace (ignored silently in MSIE)	<BS>
\f	Form Feed (ignored silently in MSIE)	<FF>
\n	Line Feed (Newline – MSIE inserts a space)	<LF>
\r	Carriage Return (MSIE inserts a space)	<CR>
\t	Horizontal Tab (MSIE inserts a space)	<HT>
\v	Vertical tab (MSIE displays the letter v)	<VT>
\0nn	Octal escape	-
\042	Double Quote	"
\047	Single Quote (Apostrophe)	'
\134	Backslash	\
\xnn	Hexadecimal escape	-
\x22	Double Quote	"
\x27	Single Quote (Apostrophe)	'
\x5C	Backslash	\
\unnnn	Unicode escape	-

Table continued on following page

Escape Sequence	Name	Symbol
\u0022	Double Quote	"
\u0027	Single Quote (Apostrophe)	'
\u005C	Backslash	\
\uFFFE	A special Unicode sentinel character for flagging byte reversed text	-
\uFFFF	A special Unicode sentinel character	-

Here are some examples of escape sequences for the apostrophe character.

In HTML apostrophe characters are escaped like this:

```
&#039;
```

In C language characters are escaped like this for octal values:

```
\047
```

And like this for hexadecimal values:

```
\x27
```

And in JavaScript, Unicode characters can be escaped like this:

```
\u0027
```

Where the `\u` is followed by a four digit hexadecimal value.

See also:	`Cast` operator, Character constant, Character display semantics, Character set, Comment, Constant, Constant expression, Lexical convention, Newline, Script Source Text, String literal

Cross-references:

ECMA 262 edition 2 – section – 2

ECMA 262 edition 2 – section – 6

ECMA 262 edition 2 – section – 7.7.3

ECMA 262 edition 3 – section – 2

ECMA 262 edition 3 – section – 6

ECMA 262 edition 3 – section – 7.7

O'Reilly *JavaScript Definitive Guide* – page – 38

escape() (Function/global)

URL escaping a text string.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	- `escape(anInputString)`	
	- `escape(anInputString, aSwitch)`	
Argument list:	*anInputString*	A string of un-escaped (normal) characters.
	aSwitch	A switch that allows plus signs to be encoded or not

The `escape()` function computes a new version of the string value it is passed. The new version has certain characters replaced with hexadecimal escape sequences.

All character codes from zero to 32 (decimal) will be escaped.

All character codes above 126 will be escaped.

The table summarizes the characters below 127 that will be escaped.

Escape	Character
%09	Tab
%0A	Line feed
%0D	Carriage return
%20	space
%21	!
%22	"
%23	#
%24	$
%25	%
%26	&
%27	'
%28	(
%29	)

Table continued on following page

Escape	Character
%3A	:
%3B	;
%3C	<
%3D	=
%3E	>
%3F	?
%5B	[
%5C	\
%5D	]
%5E	^
%60	`
%7B	{
%7C	|
%7D	}
%7E	~
%7F	Delete

For those characters that are replaced whose Unicode encoding is 0xFF or less, a two digit escape sequence of the form %xx is used. For those characters that are replaced whose Unicode character value is greater than 0xFF, a four-digit escape sequence of the form %uxxxx is used.

The encoding is partly based on the encoding described in document RFC1738. However the entire coding scheme goes beyond the scope of that RFC document.

Most non-alphanumeric characters will be escaped. All control codes are and also the upper 128 of the 255 character codes that are normally used in a web browser will be escaped too. The example script generates a table of character codes that show how they are escaped. Note that control characters will be represented by a 'missing character' box, but on some platforms special characters are mapped for display in these lower code points (0-31).

MSIE version 4 introduces Unicode support.

Netscape supports the optional second parameter to switch on the encoding of plus signs. You should place an integer 1 in this argument to activate plus encoding.

As far as ECMAScript is concerned, this is superceded in edition 3 with a set of generalized URI handling functions. The JScript 5.5 documentation refers to the `escape()` function as a deprecated feature.

Example code:

```
// Create a 0-255 lookup table of escapes
document.write("<TABLE BORDER=1>");
document.write("<TR><TH>Index</TH>");
document.write("<TH>Char</TH>");
document.write("<TH>Escape</TH></TR>");
for(ii=0; ii<256; ii++)
{
   myBinary        = ii.toString(2);
   myBinaryPadding = "00000000".substr(1,(8-myBinary.length));

   myOctal         = ii.toString(8);
   myOctalPadding  = "000".substr(1,(3-myOctal.length));

   myHex           = ii.toString(16);
   myHexPadding    = "00".substr(1,(2-myHex.length));

   myChar          = String.fromCharCode(ii);

   document.write("<TR ALIGN=RIGHT><TD>");
   document.write(ii);
   document.write("</TD><TD>");
   document.write(" "+myChar);
   document.write("</TD><TD>");
   document.write(escape(myChar));
   document.write("</TD></TR>");
}
document.write("</TABLE>");
```

See also:	`Cast` operator, `decodeURI()`, `decodeURIComponent()`, `encodeURI()`, `encodeURIComponent()`, Function property, Global object, `unescape()`, URI handling functions

Cross-references:

ECMA 262 edition 2 – section – 15.1.2.4

ECMA 262 edition 3 – section – B.2.1

ftp://ftp.isi.edu/in-notes/

Escaped JavaScript quotes in HTML (Pitfall)

Closing quotes may appear when you least expect them.

If you are generating JavaScript calls from a database, you might use one particular field as a quote delimited text parameter like this:

```
<A HREF="javascript:alert('Some text');">Click me</A>
```

Since the JavaScript call is inside a tag attribute, it is enclosed in double quotes. This means for starters that double quotes are a problem in your data if they appear.

You can get round this by enclosing any string constants in the JavaScript domain by using single quotes. JavaScript doesn't mind either. However, that means you get a problem when a single quote turns up in your data.

You might think that your output routines that get data from the database are doing very well by detecting the apostrophe and replacing it with `&#039;`, however you might still be in trouble. Likewise if you escaped the double quote by using the HTML character entity descriptor. This is because the browser unescapes the page as it is loaded. This is extremely hard to diagnose because when you view the source, you see the character entities and not the characters they become.

You must ensure that your data extraction/insertion code places the database generated items into this scenario with JavaScript escapes and not HTML escapes.

Of course everywhere else in the document you need to escape things as HTML and not JavaScript.

This all applies vice-versa by swapping the quotes around, although the escaped character entity value changes.

The example code illustrates a set of quoted strings that break and another set that work fine.

When they break, they cause a run-time error. In some cases they even prevent the status line from displaying the right thing.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Watch the status bar as you roll over the link.<BR>
<HR>
Examples of broken quotes:<BR>
<BR>
<A HREF="javascript:alert('xxx'xxx');">Test 1.1</A><BR>
<A HREF="javascript:alert('xxx&#039;xxx');">Test 1.2</A><BR>
<A HREF='javascript:alert("xxx"xxx");'>Test 1.3</A><BR>
<A HREF='javascript:alert("xxx&#034;xxx");'>Test 1.4</A><BR>
<A HREF='javascript:alert("xxx\'xxx");'>Test 1.5</A><BR>
<A HREF="javascript:alert('xxx\"xxx');">Test 1.6</A><BR>
<HR>
These work fine:<BR>
<BR>
<A HREF="javascript:alert('xxx\'xxx');">Test 2.1</A><BR>
<A HREF='javascript:alert("xxx\"xxx");'>Test 2.2</A><BR>
<A HREF="javascript:alert('xxx&#034;xxx');">Test 2.3</A><BR>
<A HREF='javascript:alert("xxx&#039;xxx");'>Test 2.4</A><BR>
<A HREF="javascript:alert('xxx\047xxx');">Test 2.5</A><BR>
<A HREF="javascript:alert('xxx\x27xxx');">Test 2.6</A><BR>
<A HREF="javascript:alert('xxx\u0027xxx');">Test 2.7</A><BR>
<A HREF="javascript:alert('xxx\042xxx');">Test 2.8</A><BR>
<A HREF="javascript:alert('xxx\x22xxx');">Test 2.9</A><BR>
<A HREF="javascript:alert('xxx\u0022xxx');">Test 2.10</A><BR>
<HR>
</BODY>
</HTML>
```

See also:	HTML Character entity, Pitfalls, String literal

Eval code (Definition)

Script source executed by an `eval()` function call.

Availability:	ECMAScript edition – 2

Eval code is the source text that is supplied to the built-in `eval()` function.

When the `eval()` function is called, it expects a string as an argument value. The contents of that string should be syntactically correct executable script source text.

On initialization, the scope chain is based on the caller's scope chain. The variable object and its '`this`' property value are also inherited. If there is no calling context, then the global object is used instead. The scope chain is identical to the caller's. Variable instantiation is performed by using the variable object belonging to the caller, but with empty property attributes. The caller provides its own value.

Eval code therefore executes very much as if it were part of the caller's execution context.

See also:	`eval()`, Executable code, Execution context, `JSObject.eval()`

Cross-references:

ECMA 262 edition 2 – section – 10.1.2

ECMA 262 edition 2 – section – 10.2.2

ECMA 262 edition 3 – section – 10.1.2

ECMA 262 edition 3 – section – 10.2.2

eval() (Function/global)

Execute some script source passed as an argument.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	Depends on the script source passed as an argument	
JavaScript syntax:	-	`eval(`*`aSourceText`*`)`
Argument list:	*`aSourceText`*	A string value containing some syntactically correct script source code.

When the `eval()` function is called, it expects a string to be passed to it as its single argument value. The contents of that string should be syntactically correct executable script source text.

The script code gets executed and any result it generates is returned. That value must be explicitly returned, otherwise the result will be undefined.

Warnings:

- If the script source passed to the `eval()` function cannot be parsed without failure, a run-time error will result.
- Be careful how you let people pass values from outside into this function. It is feasible to provide a way for a user to type in some valid JavaScript and to then execute it for them in an `eval()` function. This can be dangerous, not only because it exposes all the variables in the script but also it may be possible to construct a JavaScript that when executed, talks back to the server that provided the page in the first place.
- It would be an unusual thing to do anyway, but the possibility may be there to compromise your server security. It rather depends on the security in the hosting environment. Possibly an `eval()` action is not permitted to do things that a non-user-modifiable script embedded in a web page can do. However, this is likely to be very implementation specific.

Example code:

```
// Create some script source
var scriptCode = "c = a * b";
var a = 5;
var b = 10;
var c = 2;
document.write(c);
document.write("<BR>");
eval(scriptCode);
document.write(c);
```

See also:	Eval code, Function code, Function property, `function( ... ) ...`, Global object, `JSObject.eval()`, `Object.eval()`, `Window.setInterval()`, `Window.setTimeout()`

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 – section – 10.1.2

ECMA 262 edition 2 – section – 15.1.2.1

ECMA 262 edition 3 – section – 10.1.2

ECMA 262 edition 3 – section – 15.1.2.1

Wrox *Instant JavaScript* – page – 28

EvalError object (Object/core)

A native error object based on the `Error` object.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 Netscape – 6.0	
Inherits from:	`Error` object	
JavaScript syntax:	N	*myError* = new EvalError()
	N	*myError* = new EvalError(*aNumber*)
	N	*myError* = new EvalError(*aNumber*, *aText*)
Argument list:	*aNumber*	An error number
	aText	A text describing the error

This sub-class of the `Error` object is used when an exception is caused by using the global `eval()` function incorrectly.

See also:	`catch( ... )`, `Error` object, `RangeError` object, `ReferenceError` object, `SyntaxError` object, `throw`, `try ... catch ... finally`, `TypeError` object, `URIError` object

Inheritance chain:

`Error` object

Cross-references:

ECMA 262 edition 3 – section – 15.11.6.1

Event-driven model (Definition)

A contextual manner in which script functions are executed.

Events are generated by all kinds of occurrences. For example, an event is generated when a web page has finished loading.

Some events are triggered as a result of some user interaction, for example, clicking a mouse button or passing a mouse pointer across the screen. As the mouse crosses over an item on the screen, it generates an event signifying that the mouse has entered the region occupied by the screen item. Similarly, an event is generated when the mouse moves away from the item.

Other events can be generated by plugins as they execute some special dynamic animated or audio/visual presentation.

Sometimes these events may be useful to trap a form submission or maybe a user has entered some data into a text cell and you want to check it right away before the user does something else.

Sometimes, events might be generated on a regular basis with an interval timer.

There are other more subtle events such as errors, property changes and focus changes. Some of these are derived and are triggered as a consequence of other events.

There is an association between an event and an event handler. The handler is a JavaScript function having the same or a similar name as the event. If an event handler is not present, then the event is discarded and the browser continues interacting with the user.

See also:	`Element.onevent`, `MutationEvent` object

Event (Definition)

A trigger that may occur at any time.

Event mechanisms are good way to add functionality to the user interaction. Rather than let the browser handle the user's mouse clicks, you can intervene with JavaScript and add all kinds of sophisticated behavior to each user-driven trigger.

An event is simply a trigger generated manually by the user or as a consequence of something happening in the host environment. Events can be generated at the following times:

- When a user clicks on something
- When something finished being loaded from the web server
- When an error occurs
- When a Java exception happens
- When a timer runs out
- When a page is closed
- When a page needs to be refreshed
- When a plugin calls back to its parent page
- When a property changes while it has a `watch()` active on it
- Server-side initiation

Each of these is associated with a handler. Most of them are triggered as HTML intrinsic events.

By default, the triggers behave as if a `null` handler is assigned to the event so it appears that nothing happens when the event fires. It's likely that inside the interpreter the event is processed and simply calls a handler that returns straight away. You can replace that handler whenever you want to as the page content is processed.

MSIE version 4 implements virtually every event on every object. We describe each of those events in its own topic.

Trying to integrate both MSIE and Netscape browser event models into a single unified concept is something that is vexing the W3C experts as they deliberate over higher level functionality in the DOM. This is not likely to be resolved for some time.

Even the documentation from Netscape is inconsistent as to which event and object combinations are supported on different platforms, and you should attempt to test any portable scripts and event handlers on a variety of system configurations before deployment. Netscape 6.0 is far more standards based and consistent with MSIE as far as event handling is concerned. It implements the DOM level 2 event support. Documentation on the new version is still scarce, even on the Netscape web sites.

Draft DOM level 3 documents describing an MSIE-like event model have been published and the relevant topics are included here because that event model is supported by Netscape 6.

You may be able to establish what events are supported by an object if you write an inspection script to enumerate and print out the names of any properties of that object. You should see place holder properties for each event handler. These will be undefined unless you have registered a handler function with them.

See also:	DOM Events, Script execution

Cross-references:

Wrox *Instant JavaScript* – page – 52

event (Property)

During event handling, MSIE stores a reference to an `Event` object in this variable.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Event` object	
JavaScript syntax:	IE	`event`
	IE	*`myWindow`*`.event`

Example code:

```
// Browser portable event handler
function myEventHandler(anEvent)
{
   if(navigator.appName.indexOf("Microsoft") != -1)
   {
      anEvent = window.event;
      anEvent.srcElement = target;
      anEvent.button = which;
      anEvent.keyCode = which;
      anEvent.altKey = anEvent.modifiers & Event.ALT_MASK;
      anEvent.ctrlKey = anEvent.modifiers & Event.CONTROL_MASK;
      anEvent.shiftKey = anEvent.modifiers & Event.SHIFT_MASK;
      anEvent.clientX = pageX;
      anEvent.clientY = pageY;
   }
   // Portable event code goes here.
}
```

See also:	`Element.onevent`,`Window.event`

Property attributes:

`ReadOnly`.

Event bubbling (Definition)

A mechanism whereby MSIE and Netscape 6.0 pass events up a hierarchy of objects until they find a handler.

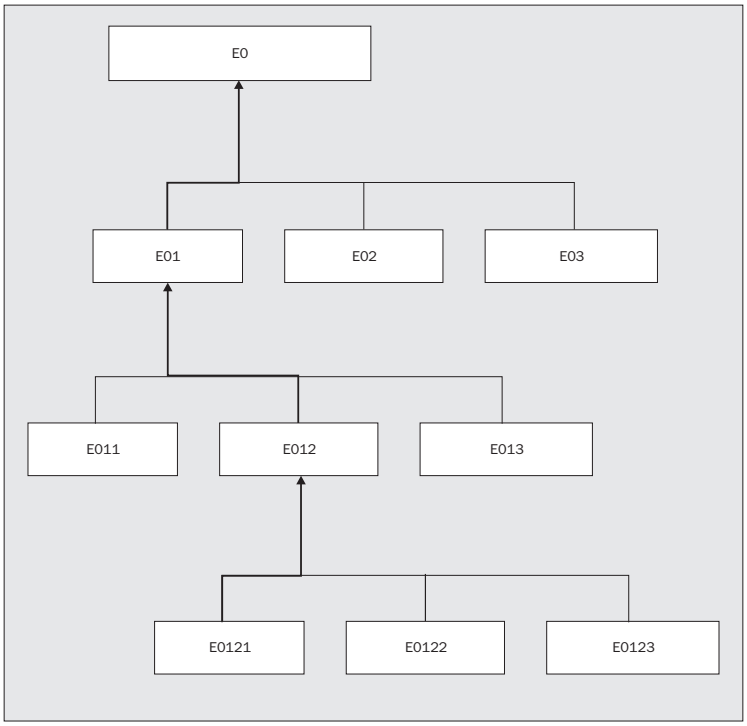
Event bubbling was introduced in version 4.0 of MSIE and provides a structured and hierarchical way of handling events. This is somewhat more object oriented in its approach than that provided by the Event Management suite in Netscape prior to version 6.

Functionally, this is very similar to the event hierarchy model found in HyperCard. Other hyper-linking and events-driven systems have modelled their event handling on this same technique since it became popularized in HyperCard in 1987.

The event bubbling technique hands an event to the most specific object relating to the event, such as the object representing an HTML tag. If there is no handler, the event is passed up the document object tree until a handler that matches is found or until it is established without doubt that there is no handler.

Handlers are able to consume an event and terminate the event handling process for that event or they may choose to pass the event upwards to a parent object. This is very sub-class, super-class like behavior and is again borrowed from the HyperCard and Object Oriented prototypes.

Event bubbling has been selected as a candidate for standardization, since it is a fundamentally more useful and automated way to propagate events through the DOM. DOM level 2 documents describing an MSIE-like event model have been published and the relevant topics are included here because that event model is supported by Netscape version 6. The event model is expected to be enhanced a little at DOM level 3 when that standard matures.



See also:	`Element.onevent`, Event, Event handler, Event management, Event model, Event object, Event propagation

Cross-references:

Wrox *Instant JavaScript* – page – 55

Event handler (Definition)

A script function that is called in response to an event trigger.

Property/method value type:	`Boolean primitive`

Each event is associated with a handler. By default a `null` handler is assigned to the event so it appears that nothing happens when the event fires. It's likely that inside the interpreter the event is processed and simply calls a handler that returns straight away. You can replace that handler as the page content is processed.

Event handlers are associated with objects in the following ways:

- By placing JavaScript code in HTML tag attributes
- By setting the object property for the handler

An event handler should conform to the following pattern:

```
function checkValue(anObject){   \\ someCode...   return true;}
```

The object that is passed to the handler is an `Event` object that describes the event that has called the handler. This was not always supported and so may be null in some implementations. It was introduced with Netscape version 4.0.

When building a page, it is sensible to place the handler functions in the `<HEAD>` block. You can do script driven property assignments to attach the handler to the objects, but that cannot take place until the page is almost complete. It is far more sensible to use the HTML tag attribute to attach the handler function where it is needed. It also keeps everything to do with that tag and its attributes in one place. The other disadvantage to script assigning the handler to a property is that the document tree needs to be walked. This can sometimes take a while and can be problematical if the `</BODY>` tag closure has not yet been processed. Legally you cannot place anything after a `</BODY>` closure. The only way then to ensure something happens when the `<BODY>` block is closed is to execute it under the `<BODY ONLOAD="...">` handler. But then why do that if you won't use tag attributes for the `<INPUT>` handlers.

If you want to, you can include the entire script of the event handler code at the point where it is associated with the HTML tag. However, this then prevents reuse and becomes hard to manage. It is better to gather the event handling logic into a single named function, organized so that it has everything it needs to process that event. Then, simply reference the function in the tag attribute that is associated with the event.

Event handlers should be designed to execute quickly and return. This is because, by their nature, they are part of the User Interface feedback process and anything that slows down the interaction between the user and the web page tends to frustrate the user and makes the application hard to use. You can let the user know that an action may take a while to complete. Even better, maybe you can create a progress indicator to show them how it's doing.

Event handlers may be invoked sooner than you expect them to be. Certainly, it is possible for them to be triggered before the page is fully loaded into the browser. Be careful that a prematurely triggered event handler does not try to manipulate objects that do not yet exist.

Event handler functions should return Boolean `true` or `false` depending on what action you want the host environment to take following your event handler's processing of the event.

Warnings:

- Event handlers should always return `true` or `false`. However, this is not enforced and won't generate an error so it isn't always necessary and often won't cause anything to happen whichever value you return. However, this might change if the interpreter vendors tighten up their functional specifications, so it's good practice to always return a Boolean result.
- Some event handlers cannot be set from a tag attribute. Those will have to be set from a script by assigning the handler to the appropriate property.
- Event handling architectures differ somewhat between Netscape and MSIE. Netscape provides event routing management while the event hierarchy allows events to bubble up through the document object model. If you are doing anything other than simply handling the event through the default handler, you need to be aware of the difference.

Example code:

```
<HTML>
<HEAD>
<SCRIPT>
function errorHandler()
{
   alert("An error has occurred");
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
window.onerror = errorHandler;
ablert("an error");
</SCRIPT>
</BODY>
</HTML>
```

See also:	Adding JavaScript to HTML, `unwatch()`, `watch()`

Cross-references:

Wrox *Instant JavaScript* – page – 52

Event handler in <SCRIPT> (Definition)

You can associate an event handler using the `<SCRIPT>` tag's attributes.

HTML syntax:	`<SCRIPT FOR="aName" EVENT="anEvent">`	
Argument list:	*aName*	The name of an `<INPUT>` element
	anEvent	The name of an event to be handled

In MSIE, you can give each `<INPUT>` tag a `NAME` attribute.

This can then be used to map a `<SCRIPT>` block to that `<INPUT>` object. The `<SCRIPT>` block is further mapped to the event names that the `<INPUT>` block might generate trigger events for. You end up with one `<SCRIPT>` block per event per named `<INPUT>` object.

The upside of this is that you don't have complicated function name mappings, but the downside is that you can end up with a large number of discrete `<SCRIPT>` blocks.

Internally, MSIE has to be turning these into some kind of function object otherwise they would be interpreted as inline Global code.

Warnings:

- Beware of using functionality that is not portable to your entire target audience. This is useful, but only on MSIE browsers. This capability is born out of the VBScript support in MSIE and is better confined to that language.

See also:	`<SCRIPT EVENT="...">`, `Element.onevent`, Script execution

Event handler properties (Definition)

Objects can have event handlers attached via property values.

From JavaScript version 1.1, you can attach event handlers by saving a reference to a function object in a property whose name is the same as the event.

If there is no handler defined then MSIE stores a reference to the `null` object in the `event` property of the `Element` object, while Netscape leaves the property undefined.

The property names must be all lower case even though the convention is to use mixed case when the event handlers are defined in HTML as tag attributes.

It is a useful thing to be able to define event handlers like this since mixing JavaScript and HTML in tag attributes is somewhat messy and can become more difficult to maintain.

A major benefit of this technique is that you can dynamically change the event handler associated with an object without changing the document's HTML. You might want to make the error handler more complex after repeated errors. Or you might want to reuse some objects and have form value checking change according to some user choices.

See also:	Calling event handlers, `Element.onevent`

Event handler scope (Definition)

The scope of an event handler is somewhat different to the normal scope.

Although event handlers are functions like any other, they are not called from global code and therefore are running in a different scope chain context to that which your functions normally execute.

The head of the scope chain is the `call` object. Arguments passed to the event handler are available here as they would be in other function calls.

In a normal context, the next object in the scope chain might be the `global` object. However for the event handler, it is the object to which the event was directed. This makes event handles somewhat easier to write.

For example, the `form` object yielded in the scope for an event handler attached to an `<INPUT>` tag is the `form` object that the `FormElement` object belongs to. This scope chain continues up the document hierarchy (the DOM), until it reaches the `global` object. You can share various parts of the form handling code by plugging it into objects at different points in the DOM.

The scope chain for an `<INPUT>` object is the reverse of its DOM location. Thus:

```
window.document.form.button.event
```

will seek to resolve its identifiers in this order:

- `call` object
- `button` object
- `form` object
- `document` object
- `window` (`global`) object

Warnings:

- You need to be explicit about how objects and event handlers are declared and called.
- Inside an event handler, you should call `window.open()` and not just `open()` because the `document` object is located sooner in the scope chain than the `window` object. This means that `open()` will call the `document.open()` method before it would have resolved to the `window.open()` method.
- You should also be careful not to add properties to your objects whose name is the same as other pre-defined objects in the DOM. If you do, you risk seriously compromising your script's ability to run correctly within its scope.
- Beware that only the code defined in the HTML tag attribute is executed in an event handler scope chain. Calls to other functions will execute them in the scope that they are defined in due to the static scope rules for functions. You can get round this by defining a JavaScript version 1.2 `Closure` object. However this technique is not supported by MSIE version 4 browsers.

See also:	Document, `Element.onevent`, Function scope, `Function.call()`

Event management (Definition)

The Netscape 4 event management suite provides event routing facilities.

The Netscape browser provides event management capabilities in version 4.0 that are different to those provided by MSIE. There are four management functions provided to facilitate this:

- `captureEvents()`
- `releaseEvents()`
- `routeEvent()`
- `handleEvent()`

Without using these management functions, events simply fire at the default handler for the event. Now you can route the events to other handlers, invoking whichever ones you need. As such, this adds to the basic functionality.

See also:	`captureEvents()`, `Document.captureEvents()`, DOM Events, `Element.onevent`, Event, Event bubbling, Event handler, Event model, `handleEvent()`, `MutationEvent` object, `unwatch()`, `watch()`, `Window.routeEvent()`

Cross-references:

Wrox *Instant JavaScript* – page – 55

Event model (Definition)

A framework within which events are triggered and handled by the interpreter.

The HTML 4.0 standard refers to events relating to HTML as intrinsic events. That includes the following:

- Conditions that cause events
- Attributes in tags that support events
- Scripts in the page that handle events
- A JavaScript host object that an event operates on
- Control being passed temporarily to the interpreter during the event handling
- Data associated with and passed to the handler by the event

The DOM level 2 standard introduces an event model based on the MSIE event bubbling technique. It is embodied in the following object classes:

- `EventTarget`
- `EventListener`
- `Event`
- `EventException`
- `DocumentEvent`
- `UIEvent`
- `MouseEvent`

- `MutationEvent`

Events can be added using the `addEventListener()` method that is inherited by any Elements that can be treated as `EventTarget` objects. The `EventTarget` class is not a concrete class in its own right but is an extension of an existing object class to add event handling capabilities to it.

DOM level 3 is expected to enhance the event model further with the addition of the following classes:

- `KeyEvent`
- `EventGroup`
- `EventGrouped`
- `DocumentEventGroup`

See also:	DOM Events, Dynamic HTML, `Element.onevent`, `MutationEvent` object, `unwatch()`, `watch()`

Event names (Definition)

Rules for naming event handlers.

When you specify the name of an event as an HTML tag attribute, because tag attributes in HTML are case insensitive, you only have to spell the event name correctly. You can type it in any mixture of upper and lower case and the browser should still be able to associate it with the correct fragment of JavaScript. These are all equally valid:

- `<INPUT TYPE="text" ONCHANGE="handleEvent();">`
- `<INPUT TYPE="text" onchange="handleEvent();">`
- `<INPUT TYPE="text" onChange="handleEvent();">`
- `<INPUT TYPE="text" OnChAnGe="handleEvent();">`

It is conventional to do it like this:

- `<INPUT TYPE="text" onChange="handleEvent();">`

Inside the JavaScript interpreter, you can associate handlers using properties. However the event names must be correctly cased in that situation: they would be lower case. You don't need to put the `onChange` tag attribute in, because you are registering the error handler inside the script block, like this:

```
<HTML>
<BODY>
<FORM NAME="the_form"><INPUT TYPE="text" NAME="the_text"></FORM>
<SCRIPT>document.the_form.the_text.onchange=alert("Changed");</SCRIPT>
</BODY>
</HTML>
```

Here is a list of all event names although they don't all apply to every object in the document.

Event	Handler	Usage
Abort	`onabort`	When image loading is aborted.
AfterPrint	`onafterprint`	When printing has just finished.
AfterUpdate	`onafterupdate`	When an update is completed.

Table continued on following page

Event	Handler	Usage
Back	`onback`	The user has clicked on the [BACK] button in the toolbar.
BeforeCopy	`onbeforecopy`	Immediately before a copy to the clipboard.
BeforeCut	`onbeforecut`	Immediately before a cut to the clipboard.
BeforeEditFocus	`onbeforeeditfocus`	Immediately before the edit focus is directed to an element.
BeforePaste	`onbeforepaste`	Immediately before the clipboard is pasted.
BeforePrint	`onbeforeprint`	Immediately before printing begins.
BeforeUnload	`onbeforeunload`	Called immediately prior to the window being unloaded.
BeforeUpdate	`onbeforeupdate`	Called immediately before an update commences.
Blur	`onblur`	When an input element loses input focus.
Bounce	`onbounce`	Triggered when a marquee element hits the edge of its element area.
Change	`onchange`	When edit fields have new values entered or a popup has a new selection, this event's handler can check the new value.
Click	`onclick`	When the user clicks the mouse button on the `Element` object that represents the object on screen.
ContextMenu	`oncontextmenu`	Special handling for contextual menus.
Copy	`oncopy`	When a copy operation happens.
Cut	`oncut`	When a cut operation happens.
DataAvailable	`ondataavailable`	Some data has arrived asynchronously from an applet or data source.
DataSetChanged	`ondatasetchanged`	A data source has changed the content or some initial data is now ready for collection.
DataSetComplete	`ondatasetcomplete`	There is no more data to be transmitted from the data source.
DblClick	`ondblclick`	When the user double-clicks on an object.
Drag	`ondrag`	When a drag operation happens.
DragDrop	`ondragdrop`	Some data has been dropped onto a window.
DragEnd	`ondragend`	When a drag ends.
DragEnter	`ondragenter`	When a dragged item enters the element.
DragLeave	`ondragleave`	When a dragged item leaves the element.
DragOver	`ondragover`	While the dragged item is over the element.
DragStart	`ondragstart`	The user has commenced some data selection with a mouse drag.
Drop	`ondrop`	When a dragged item is dropped.
Error	`onerror`	Triggered if an error occurs when loading an image.
ErrorUpdate	`onerrorupdate`	An error has occurred in the transfer of some data from a data source.

Table continued on following page

Event	Handler	Usage
FilterChange	`onfilterchange`	A filter has changed the state of an element or a transition has just been completed.
Finish	`onfinish`	A marquee object has finished looping.
Focus	`onfocus`	When the form element is selected for entry.
Forward	`onforward`	The user has clicked on the [FORWARD] button in the toolbar.
Help	`onhelp`	The user has pressed the [F1] key or selected [help] from the toolbar or menu.
KeyDown	`onkeydown`	When a key is pressed.
KeyPress	`onkeypress`	Pressing the key down and releasing it again elicits this event.
KeyUp	`onkeyup`	When a key is released.
Load	`onload`	When an object has completed loading.
LoseCapture	`onlosecapture`	When an element loses event capturing permission.
MouseDown	`onmousedown`	When the mouse button is pressed.
MouseDrag	`onmousedrag`	An event handler for mouse drag operations.
MouseMove	`onmousemove`	When the mouse pointer is moved.
MouseOut	`onmouseout`	When the mouse pointer leaves the active area occupied by the `Element` object that represents the object on screen.
MouseOver	`onmouseover`	When the mouse pointer enters the active area owned by the object.
MouseUp	`onmouseup`	When the mouse button is released.
Move	`onmove`	The browser window has been moved.
Paste	`onpaste`	When a paste operation happens.
PropertyChange	`onpropertychange`	When an object property is modified (similar to the Netscape Navigator `watch()` method).
ReadyStateChange	`onreadystatechange`	An object in the window has changed its ready state.
Reset	`onreset`	The user has clicked a reset button in a form.
Resize	`onresize`	As the window is resized, this event is triggered.
RowEnter	`onrowenter`	The data in a field bound to a data source is about to be changed.
RowExit	`onrowexit`	The data in a field bound to a data source has been changed.
Scroll	`onscroll`	The window has been scrolled.
Select	`onselect`	Some textual content in the window has been selected.
SelectStart	`Onselectstart`	A select action is beginning.
Start	`Onstart`	A marquee element is beginning its loop.

Table continued on following page

Event	Handler	Usage
Stop	`Onstop`	When a stop action occurs.
Submit	`Onsubmit`	The user has clicked on the submit button in a form.
Unload	`onunload`	Triggered when the document is unloaded.

In the topics that describe the events, the intercap (capitalised word breaks) form is used so that the event name reads more easily.

Warnings:

- Netscape version 4.0 allows event name properties to have some uppercase characters. For example, a portable script would set the `onchange` property. A non-portable script that only worked in Netscape would set the `onChange` property. It is probably best to avoid using this if you want your scripts to run as widely as possible.
- Not all versions of the browsers support the assignment of handlers to object properties. This depends on the object type, browser version and probably the phase of the moon.

See also:	Event, Event handler, Event model, Keyboard events, `Window.releaseEvents()`

Event object (Object/DOM)

Early Netscape and MSIE browsers define different event object models. In MSIE, a single `Event` object is available globally and shared by all events.

Availability:	DOM level – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
JavaScript syntax:	-	*myEvent* = event
	-	*myEvent* = *myWindow*.event
	N	*myEvent* = *myDocumentEvent*.createEvent(*aType*)
Argument list:	*aType*	An event type
Object properties:	altKey, bubbles, button, cancelable, cancelBubble, charCode, clientX, clientY, ctrlKey, currentTarget, data, dataFld, dataTransfer, eventPhase, fromElement, height, keyCode, layerX, layerY, modifiers, offsetX, offsetY, pageX, pageY, propertyName, qualifier, reason, recordset, repeat, returnValue, screenX, screenY, shiftKey, srcElement, srcFilter, srcUrn, target, timeStamp, toElement, type, which, width, x, y	

Class constants:	ABORT, ALT_MASK, AT_TARGET, BACK, BLUR, BUBBLING_PHASE, CAPTURING_PHASE, CHANGE, CLICK, CONTROL_MASK, DBLCLICK, DRAGDROP, ERROR, FOCUS, FORWARD, HELP, KEYDOWN, KEYPRESS, KEYUP, LOAD, LOCATE, META_MASK, MOUSEDOWN, MOUSEDRAG, MOUSEMOVE, MOUSEOUT, MOUSEOVER, MOUSEUP, MOVE, RESET, RESIZE, SCROLL, SELECT, SHIFT_MASK, SUBMIT, UNLOAD, XFER_DONE
Object methods:	initEvent(),preventDefault(),stopPropagation()
Collections:	bookmarks[],boundElements[]

The event models in Netscape version 4 and MSIE version 4 support an `Event` object. However, both browsers support different properties for this object. It may be possible with some smart JavaScript code to normalize these to look like the same object model. You could either define your own or take one and emulate it in the other.

As well as being different, the `Event` object is passed to event handlers in a different way for each browser. Netscape version 4 passes the `Event` object as an argument. MSIE version 4 stores a reference to it in the global variable called `event`.

Netscape provides a set of static constants that can be used to manufacture modifier masks. These can then be tested against the `Event.modifiers` property. These constants have names that all end with the suffix "`_MASK`". There are other constants provided so masks for event types can be made. These can be tested against the `Event.type` property.

MSIE does not support these static constants in the same way and uses fully spelled-out event names as string primitive values for matching. This ultimately may be better because Netscape is constrained for space as regards bit values for new events. There are only 32 bits and most have already been allocated to event types.

As of version 6.0 of Netscape, the underlying event model is based on the DOM level 2 event module. This is fundamentally different to the support of events in earlier versions of the Netscape browser. DOM level 2 and Netscape 6.0 converge on the same basic model as MSIE. This adds some new properties, methods and static constants in accordance with the DOM specification. Some properties persist from earlier versions.

The DOM level 2 specification for events describes a category of HTMLEvents which it suggests is based on DOM level 0 capabilities. There is only a small amount of information about the event type strings and the event capabilities in the DOM level 2 context. No ECMAScript binding is described and because DOM level 0 has never been published as a standard, there is a little ambiguity about these events. The standardization is somewhat de-facto regarding them and in due course, as the DOM event model evolves, this should all become more consistent and more completely documented.

Warnings:

- Properties belonging to an `Event` object cannot be set in Netscape unless the script has the `UniversalBrowserWrite` privilege. You also cannot watch events in other windows if they are loaded from different sources unless the script also has the `UniversalBrowserWrite` privilege.

- The properties and methods supported by Netscape and MSIE differ greatly in name and function. You will need to be very careful when building sophisticated event management capabilities into your scripts.
- Netscape does not implement all of the events for which there are static constants defined.

Example code:

```
// List the event constants in Netscape Navigator.
// This does not work in MSIE.
d  = document;
e  = Event;
s1 = "<TR><TD>";
s2 = "</TD><TD>";
s3 = "</TD></TR>";
d.write("<TABLE BORDER=1>");
d.write(s1 + "MOUSEDOWN" + s2 + e.MOUSEDOWN + s3);
d.write(s1 + "MOUSEUP"   + s2 + e.MOUSEUP   + s3);
d.write(s1 + "MOUSEOVER" + s2 + e.MOUSEOVER + s3);
d.write(s1 + "MOUSEOUT"  + s2 + e.MOUSEOUT  + s3);
d.write(s1 + "MOUSEMOVE" + s2 + e.MOUSEMOVE + s3);
d.write(s1 + "CLICK"     + s2 + e.CLICK     + s3);
d.write(s1 + "DBLCLICK"  + s2 + e.DBLCLICK  + s3);
d.write(s1 + "KEYDOWN"   + s2 + e.KEYDOWN   + s3);
d.write(s1 + "KEYUP"     + s2 + e.KEYUP     + s3);
d.write(s1 + "KEYPRESS"  + s2 + e.KEYPRESS  + s3);
d.write(s1 + "DRAGDROP"  + s2 + e.DRAGDROP  + s3);
d.write(s1 + "FOCUS"     + s2 + e.FOCUS     + s3);
d.write(s1 + "BLUR"      + s2 + e.BLUR      + s3);
d.write(s1 + "SELECT"    + s2 + e.SELECT    + s3);
d.write(s1 + "CHANGE"    + s2 + e.CHANGE    + s3);
d.write(s1 + "RESET"     + s2 + e.RESET     + s3);
d.write(s1 + "SUBMIT"    + s2 + e.SUBMIT    + s3);
d.write(s1 + "LOAD"      + s2 + e.LOAD      + s3);
d.write(s1 + "UNLOAD"    + s2 + e.UNLOAD    + s3);
d.write(s1 + "ABORT"     + s2 + e.ABORT     + s3);
d.write(s1 + "ERROR"     + s2 + e.ERROR     + s3);
d.write(s1 + "MOVE"      + s2 + e.MOVE      + s3);
d.write(s1 + "RESIZE"    + s2 + e.RESIZE    + s3);
d.write("</TABLE>");
```

See also: `Element.onevent`, Event bubbling, Event handler, Event type constants, `Event.modifiers`, `Implementation.hasFeature()`, `MouseEvent` object, Timer events, `UIEvent` object, `UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`, `unwatch()`, `watch()`, `Window.handleEvent()`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`altKey`	-	3.0 +	-	4.0 +	5.0 +	-	ReadOnly
`bubbles`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`button`	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly
`cancelable`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
cancelBubble	-	3.0 +	-	4.0 +	5.0 +	-	-
charCode	-	3.0 +	-	4.0 +	-	-	-
clientX	-	3.0 +	-	4.0 +	5.0 +	-	ReadOnly
clientY	-	3.0 +	-	4.0 +	5.0 +	-	ReadOnly
ctrlKey	-	3.0 +	-	4.0 +	5.0 +	-	ReadOnly
currentTarget	1.5 +	-	6.0 +	-	5.0 +	2 +	ReadOnly
data	1.2 +	3.0 +	4.0 +	4.0 +	-	-	Warning, ReadOnl.
dataFld	1.2 +	3.0 +	4.0 +	4.0 +	-	2 +	Warning
dataTransfer	-	5.0 +	-	5.0 +	-	-	-
eventPhase	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
fromElement	-	3.0 +	-	4.0 +	-	-	ReadOnly
height	1.2 +	-	4.0 +	-	-	-	-
keyCode	-	3.0 +	-	4.0 +	-	-	Warning
layerX	1.2 +	-	4.0 +	-	-	-	Warning, ReadOnly
layerY	1.2 +	-	4.0 +	-	-	-	Warning, ReadOnly
modifiers	1.2 +	-	4.0 +	-	-	-	ReadOnly
offsetX	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly
offsetY	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly
pageX	1.2 +	-	4.0 +	-	-	-	ReadOnly
pageY	1.2 +	-	4.0 +	-	-	-	ReadOnly
propertyName	-	5.0 +	-	5.0 +	-	-	-
qualifier	1.2 +	3.0 +	4.0 +	4.0 +	-	2 +	Warning
reason	-	3.0 +	-	4.0 +	-	-	ReadOnly
recordset	1.2 +	3.0 +	4.0 +	4.0 +	-	2 +	Warning
repeat	-	5.0 +	-	5.0 +	-	-	-
returnValue	-	3.0 +	-	4.0 +	-	-	-
screenX	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	-	ReadOnly
screenY	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	-	ReadOnly
shiftKey	-	3.0 +	-	4.0 +	5.0 +	-	ReadOnly
srcElement	-	3.0 +	-	4.0 +	-	-	ReadOnly
srcFilter	-	3.0 +	-	4.0 +	-	-	ReadOnly
srcUrn	1.2 +	3.0 +	4.0 +	4.0 +	-	2 +	Warning
target	1.2 +	-	4.0 +	-	5.0 +	2 +	ReadOnly

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
timeStamp	1.5 +	-	6.0 +	-	-	2 +	Warning, ReadOnly
toElement	-	3.0 +	-	4.0 +	-	-	ReadOnly
type	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	2 +	ReadOnly
which	1.2 +	-	4.0 +	-	5.0 +	-	Warning, ReadOnly
width	1.2 +	-	4.0 +	-	-	-	-
x	1.2 +	3.0 +	4.0 +	4.0 +	-	-	ReadOnly
y	1.2 +	3.0 +	4.0 +	4.0 +	-	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
initEvent()	1.5 +	-	6.0 +	-	-	2 +	-
preventDefault()	1.5 +	-	6.0 +	-	5.0 +	2 +	-
stopPropagation()	1.5 +	-	6.0 +	-	5.0 +	2 +	-

Event.altKey (Property)

A Boolean value in MSIE that represents the state of the [alt] key.

Availability:	JScript – 3.0 Internet Explorer version – 4.0 Opera – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myEvent*.altKey

When an event is being processed, you may want to know the state of the *[alt]* key on the keyboard.

This Boolean property returns true when the *[alt]* key is pressed and false when it is not.

This property reflects the state of the *[alt]* key at the instant when the event was triggered. The user may have released the *[alt]* key in the meantime, so you should not assume that if the *[alt]* key was pressed earlier on that it is still pressed when the event handler is being executed.

DOM level 2 event handling moves this property to the `MouseEvent` object.

See also:	`Event.modifiers`, `MouseEvent.altKey`, `onKeyDown`

Property attributes:

`ReadOnly`.

Event.bookmarks[] (Collection)

A collection of all the ADO bookmarks tied to the rows affected by the current event.

JavaScript syntax:	ASP	*myEvent*.bookmarks

This is part of the ADO server side support in the Microsoft ASP server.

See also:	Active Server Pages, ADO

Property attributes:

`ReadOnly`.

Event.boundElements[] (Collection)

A collection of all the elements on the page that are bound to a dataset.

JavaScript syntax:	ASP	*myEvent*.boundElements

This is part of the ADO server side support in the Microsoft ASP server.

See also:	Active Server Pages

Property attributes:

`ReadOnly`.

Event.bubbles (Property)

A Boolean value that indicates whether the event can bubble or not.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	N *myEvent*.bubbles

If this value is `true`, then the event can bubble. That means it can propagate upwards to its parent node and onwards towards the head of the document hierarchy.

Property attributes:

`ReadOnly`.

Event.button (Property)

The mouse button that was pressed to trigger the event in MSIE.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myEvent*`.button`

This property will contain a value to indicate the mouse button that was pressed to trigger the event.

Note that on a Macintosh, this value is meaningless as there is only one button on an Apple Macintosh mouse although you can retrofit a third party 2 or 3 button mouse.

Third party mice can be added to Apple Macintosh systems to provide multiple (2 or 3) button functionality, but they are by no means commonplace and it is probably safe to assume only a 1 button mouse on the Macintosh platform.

The values for the buttons are defined to cope with 1, 2 or 3 button mice.

The following values correspond with mouse buttons:

- 0 – No mouse button was pressed when the event was triggered
- 1 – The left button was pressed
- 2 – The right button was pressed
- 3 – The middle button was pressed

DOM level 2 event handling moves this property to the `MouseEvent` object.

Warnings:

- This is not supported by Netscape which returns mouse button values and key codes in the same property (`which`).

See also:	`Event.which`, `MouseEvent.button`, `onClick`, `onDblClick`, `onMouseDown`, `onMouseUp`

Property attributes:

`ReadOnly`.

Event.cancelable (Property)

If the event can be cancelled, then this flag will be set `true`.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*`myEvent`*`.cancelable`

Events can be cancelled and have their default action prevented. If the event can support this, then the `true` value will be present. This will affect the value returned to the dispatcher that triggered the event. Dispatchers should behave consistently whether they are `EventTarget` objects or the implementation that the scripts are executing in.

Property attributes:

`ReadOnly`.

Event.cancelBubble (Property)

A flag that halts event bubbling in MSIE browsers.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myEvent`*`.cancelBubble = `*`aBooleanValue`*
Argument list:	*`aBooleanValue`*	A `true` or `false` setting

Event propagation passes events up the DOM hierarchy, calling the handler for each event in turn. This propagation effect can be stopped at once by setting the `cancelBubble` property to `true`.

Setting this property `true` in an event handler may improve stability in dynamically generated HTML fragments that have mouse rollover code associated with them.

For example, you can implement a ticker in JavaScript instead of Java. However, on some MSIE implementations (Macintosh for example), the browser crashes as you rollover several objects that have been created dynamically. Using the `cancelBubble` technique may stop runaway events from propagating and causing contention in the browser core.

See also:	Event propagation

Event.charCode (Property)

The character code of a key that was pressed to trigger the event relating to this object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myEvent*`.charCode`

Events may be triggered by keystrokes. This property provides a means of determining which key was pressed.

Note that this is a character code property so it is a character value mapped to a keyboard key and therefore should take account of internalization.

Event.clientX (Property)

MSIE mouse position relative to the web page.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myEvent*`.clientX`

This is the horizontal position of the mouse when the event was triggered. The position is calculated relative to the visible document area within the window or frame the mouse was positioned in when the event triggered.

DOM level 2 event handling moves this property to the `MouseEvent` object.

See also:	`Event.clientY`, `Event.pageX`, `Event.pageY`, `Event.screenX`, `Event.screenY`, `MouseEvent.clientX`

Property attributes:

`ReadOnly`.

Event.clientY (Property)

MSIE mouse position relative to the web page.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myEvent`*`.clientY`

This is the vertical position of the mouse when the event was triggered. The position is calculated relative to the visible document area within the window or frame the mouse was positioned in when the event triggered.

DOM level 2 event handling moves this property to the `MouseEvent` object.

See also:	`Event.clientX`, `Event.pageX`, `Event.pageY`, `Event.screenX`, `Event.screenY`, `MouseEvent.clientY`

Property attributes:

`ReadOnly`.

Event.ctrlKey (Property)

A Boolean value in MSIE that represents the state of the [control] key.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myEvent`*`.ctrlKey`

When an event is being processed, you may want to know the state of the *[control]* key on the keyboard.

This Boolean property returns true when the *[control]* key is pressed and false when it is not.

This property reflects the state of the *[control]* key at the instant when the event was triggered. The user may have released the key in the meantime, so you should not assume that if the *[control]* key was pressed earlier on that it is still pressed when the event handler is being executed.

DOM level 2 event handling moves this property to the `MouseEvent` object.

See also:	`Event.modifiers`, `MouseEvent.ctrlKey`, `onKeyDown`

Property attributes:

`ReadOnly`.

Event.currentTarget (Property)

A reference to the object whose listener is being called with the receiving `Event` object.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`EventTarget` object	
JavaScript syntax:	N	*myEvent*`.currentTarget`

This is a useful property which provides a means of sharing listener scripts between several event handlers. You can then determine which one of several objects was the `EventTarget` that had the event dispatched to it.

Property attributes:

`ReadOnly`.

Event.data (Property)

The URL of the data dropped into a the window from an `onDragDrop` event.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myEvent*`.data`

This property contains information that describes what was just dropped onto the object that owns the `onDragDrop` handler. Normally, that would be a URL that refers to the entity that was dropped.

Warnings:

- When using this property in Netscape, you will need the `UniversalBrowserRead` privilege to see the value and `UniversalBrowserWrite` privilege to change it.

See also:	`UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`, `Window.ondragdrop`

Property attributes:

`ReadOnly`.

Event.dataTransfer (Property)

A means of transferring drag and drop data via the `Event` object. This refers to a `dataTransfer` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`dataTransfer` object	
JavaScript syntax:	IE	*`myEvent`*`.dataTransfer`

The browser loads the object as it handles the drag and drop operation. Refer to the `dataTransfer` object topic for further details.

See also:	`dataTransfer` object

Event.eventPhase (Property)

Describes what phase the event is currently being processed in.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myEvent`*`.eventPhase`

A new concept that has been introduced at DOM level 2 is that of event phases. This allows your handler to determine the current disposition of an event from the script interface. You can examine the `eventPhase` property and test it against one of the predefined constants.

DOM level 2 introduces these static constants to the `Event` class to describe the current state of an event that is encapsulated by the `Event` object:

Value	Symbolic Name
1	CAPTURING_PHASE
2	AT_TARGET
3	BUBBLING_PHASE

See also:	Event type constants

Property attributes:

`ReadOnly`.

Event.fromElement (Property)

The object that the mouse is moving from.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Object` object	
JavaScript syntax:	IE	*myEvent*.fromElement

This property is useful in `onMouseOver` event handlers. It is a reference to the object that mouse was previously over when it entered an object. That object receives a `onMouseOver` event. It also knows about the `onMouseOver` object via the `toElement` property of the `Event` object.

See also:	`Event.toElement`

Property attributes:

`ReadOnly`.

Event.height (Property)

The new height of a resized window or frame.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myEvent*.height

If a window or frame is resized by the user, the new height value is available in this property.

See also:	`Event.width`

Event.initEvent() (Method)

An `Event` object initializer that must be called before dispatching the `Event` object to an `EventTarget`.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myEvent*.initEvent(*aType*, *aBubble*, *aCancel*)

Argument list:	*aType*	A string indicating the event type
	aBubble	A boolean flag indicating whether the event can bubble
	aCancel	A boolean flag indicating whether the event can be cancelled

This method is used to initialize `Event` objects that have been created with the `DocumentEvent` object. It is intended that this be called at the outset although the DOM level 2 specification allows for an implementation to call it several times later on if necessary.

The event type argument value must conform to the conventions discussed in the `Event.type` topic.

The first of the remaining arguments is a flag value to indicate whether the event is bubbling or capturing in its behavior. The last argument is a flag to indicate whether the event can be cancelled.

See also:	`Event.type`

Event.keyCode (Property)

The code point for the key that was pressed in MSIE.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myEvent*`.keyCode`

This is the Unicode code point value and not the character code. This means the value is numeric.

You can also use the mask values to determine if any of the modifier keys were held down. Here is a list of the modifier key mask values:

Bit	Constant	Event type
2^0	ALT_MASK	The [alt] key was held down while key was pressed
2^1	CONTROL_MASK	The [control] key was held down while key was pressed
2^2	SHIFT_MASK	The [shift] key was held down while key was pressed
2^3	META_MASK	The [meta] key was held down while key was pressed

Warnings:

- This is not supported by Netscape which returns choice values in the `which` property value.

See also:	`Event.which`, `onKeyDown`

Event.layerX (Property)

The X coordinate of the event within a layer.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myEvent*`.layerX`

In Netscape, mouse positions are available within a `layer` object. These may not be the same positions as the mouse coordinates within a window. This is because a layer may have been scrolled with respect to the window. Nevertheless if you are working on the layer, you need coordinates that are relative to your target to save having to compute them.

Warnings:

- Netscape 6.0 completely removes layer support. If you use layers, your pages will break.

See also:	`Event.layerY`

Property attributes:

`ReadOnly`.

Event.layerY (Property)

The Y coordinate of the event within a layer.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myEvent*`.layerY`

In Netscape, mouse positions are available within a `layer` object. These may not be the same positions as the mouse coordinates within a window. This is because a layer may have been scrolled with respect to the window. Nevertheless if you are working on the layer, you need coordinates that are relative to your target to save having to compute them.

Warnings:

- Netscape version 6.0 completely removes layer support. If you use layers, your pages will break.

See also:	`Event.layerX`

Property attributes:

`ReadOnly`.

Event.modifiers (Property)

A bitmask provided by Netscape, which contains a bit flag for each modifier key.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myEvent`*`.modifiers`

The modifiers are supplied as a mask which needs to be assembled using bitwise OR techniques from the component values.

The mask values are summarized in this table:

Bit	Constant	Event type
2^0	ALT_MASK	The [alt] key was held down while key was pressed
2^1	CONTROL_MASK	The [control] key was held down while key was pressed
2^2	SHIFT_MASK	The [shift] key was held down while key was pressed
2^3	META_MASK	The [meta] key was held down while key was pressed

See also:	`Document.releaseEvents()`, `Event` object, `Event.altKey`, `Event.ctrlKey`, `Event.shiftKey`, Keyboard events, `Layer.releaseEvents()`, `onKeyDown`, `Window.releaseEvents()`

Property attributes:

`ReadOnly`.

Event.offsetX (Property)

The X coordinate of the event relative to its containing object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myEvent`*`.offsetX`

As you would expect, coordinate positions of the mouse are provided in mutually exclusive ways in Netscape and MSIE. This is the MSIE equivalent to the layer coordinate scheme in Netscape. Of course, MSIE does not have layers but they can be simulated by using arbitrary objects.

Warnings:

- The value in this property is calculated relative the coordinate system of the container of the source element that generated the event. This may not be the same as the coordinate scheme for the receiving object.

Property attributes:

`ReadOnly`.

Event.offsetY (Property)

The Y coordinate of the event relative to its containing object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myEvent`*`.offsetY`

As you would expect, coordinate positions of the mouse are provided in mutually exclusive ways in Netscape and MSIE. This is the MSIE equivalent to the layer coordinate scheme in Netscape. Of course, MSIE does not have layers but they can be simulated by using arbitrary objects.

Warnings:

- The value in this property is calculated relative the coordinate system of the container of the source element that generated the event. This may not be the same as the coordinate scheme for the receiving object.

Property attributes:

`ReadOnly`.

Event.pageX (Property)

Netscape mouse position relative to the web page.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myEvent`*`.pageX`

This is the Netscape mechanism for locating the mouse with respect to the window or frame coordinate address space. Of course it's different to the MSIE property that provides the same functionality. This means you need to build completely different event handling scripts for each platform, as if you didn't already have reason enough.

See also:	`Event.clientX`, `Event.clientY`, `Event.pageY`, `Event.screenX`, `Event.screenY`

Property attributes:

`ReadOnly`.

Event.pageY (Property)

Netscape mouse position relative to the web page.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myEvent`*`.pageY`

This is the Netscape mechanism for locating the mouse with respect to the window or frame coordinate address space. Of course it's different to the MSIE property that provides the same functionality. This means you need to build completely different event handling scripts for each platform, as if you didn't already have reason enough.

See also:	`Event.clientX`, `Event.clientY`, `Event.pageX`, `Event.screenX`, `Event.screenY`

Property attributes:

`ReadOnly`.

Event.preventDefault() (Method)

A means of preventing the default behavior from being activated after the event returns to its dispatcher.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0 Opera – 5.0	
JavaScript syntax:	N	*`myEvent`*`.preventDefault()`

This method only applies if the event can be cancelled. That is, if the `Event` object's `cancelable` property contains a true value. This value is readable but not writable. It is defined when the `Event` object is instantiated or initialized and cannot be changed later.

Calling this method at any time during the handling of the event will cancel the event immediately, returning control to the dispatcher. Any default handling will also be cancelled. You should be aware that if you dispatch an event from a script, it is your responsibility to make sure you check the return status of your dispatch call to ensure you do this consistently with implementation dispatched events.

Calling this method should cancel any subsequent event processing regardless of the kind of propagation being used. Any subsequent bubbling up or capturing should see that the event has been cancelled and should honor that without any further processing of the event.

Calling this method on an event whose `cancelable` property does not contain a `true` value should have no effect at all.

Event.propertyName (Property)

The name of a property that was changed and which triggered an `onPropertyChange` event.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myEvent`*`.propertyName`

This property is useful in the event handler code for the `onPropertyChange` event. You can also inspect the `srcElement` property of the same `Event` object to establish the object owning the changed property.

See also:	`Event.srcElement,onPropertyChange`

Event.reason (Property)

An indication of the status of a data transfer.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myEvent`*`.reason`

This is part of the database driven event handling which is triggered by a `DataSetComplete` event.

You can check this property to determine if the data transfer worked. The following values are supported:

- 0 – The transfer was successful
- 1 – The transfer was aborted prematurely
- 2 – Some kind of error occurred during the transfer

See also:	`onDataSetComplete`

Property attributes:

`ReadOnly`.

Event.repeat (Property)

If a keyboard can generate auto-repeating keystrokes, then this is set `true` when a keystroke is an auto-repeat of a previous one.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myEvent`*`.repeat`

Refer to:

`Event.keyCode`

Event.returnValue (Property)

A return value for the event.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myEvent`*`.returnValue`

You may want to modify the return value for the event handler. By assigning a new value to this property, the `returnValue` property of the event handler will be modified.

Because event handlers can only return a Boolean value, you only need to consider the possibility of changing the default `true` value to `false`.

If you return a `true` value, the event is handed back into the browser and it is able to continue handling the user interaction. For example, if a Click handler is introduced, it would behave as if the Click handler were not present.

Returning a `false` value informs the browser that your handler has done everything that is necessary to manage the event and it can now dispose of the event without any further action being taken on its part.

Event.screenX (Property)

Mouse position relative to the screen display.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0
Property/method value type:	`Number primitive`

JavaScript syntax:	-	*myEvent*.screenX

You may need to know the position of the mouse relative to the screen display coordinates and not the browser window or objects within it. This property provides the horizontal coordinate of the mouse within the screen.

DOM level 2 event handling moves this property to the `MouseEvent` object.

See also:	`Event.clientX`, `Event.clientY`, `Event.pageX`, `Event.pageY`, `Event.screenY`, `MouseEvent.screenX`

Property attributes:

`ReadOnly`.

Event.screenY (Property)

Mouse position relative to the screen display.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myEvent*.screenY

You may need to know the position of the mouse relative to the screen display coordinates and not the browser window or objects within it. This property provides the vertical coordinate of the mouse within the screen.

DOM level 2 event handling moves this property to the `MouseEvent` object.

See also:	`Event.clientX`, `Event.clientY`, `Event.pageX`, `Event.pageY`, `Event.screenX`, `MouseEvent.screenY`

Property attributes:

`ReadOnly`.

Event.shiftKey (Property)

A Boolean value in MSIE that represents the state of the *[shift]* key.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0

Property/method value type:	Boolean primitive	
JavaScript syntax:	IE	*myEvent*.shiftKey

When an event is being processed, you may want to know the state of the *[shift]* key on the keyboard.

This Boolean property returns `true` when the *[shift]* key is pressed and `false` when it is not.

This property reflects the state of the *[shift]* key at the instant when the event was triggered. The user may have released the key in the meantime, so you should not assume that if the *[shift]* key was pressed earlier on that it is still pressed when the event handler is being executed.

DOM level 2 event handling moves this property to the `MouseEvent` object.

See also:	Event.modifiers, MouseEvent.shiftKey, onKeyDown

Property attributes:

`ReadOnly`.

Event.srcElement (Property)

An MSIE supported property containing the event source.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	Object reference	
JavaScript syntax:	IE	*myEvent*.srcElement

You can construct an event handler that is shared amongst several objects. If that is the case, this property allows you to determine which one triggered the event so that you can modify the event handler behavior accordingly.

See also:	Event.propertyName, Event.target, onPropertyChange

Property attributes:

`ReadOnly`.

Event.srcFilter (Property)

A filter object representing the filter that changed.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Property/method value type:	`Filter object reference`	
JavaScript syntax:	IE	*myEvent*`.srcFilter`

You can construct an event handler that is shared amongst several filters in MSIE. If that is the case, this property allows you to determine which one triggered the event, so that you can modify the event handler behavior accordingly.

Property attributes:

`ReadOnly`.

Event.stopPropagation() (Method)

Prevents propagation of event handling via bubbling or capture techniques.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0 Opera – 5.0	
JavaScript syntax:	N	*myEvent*`.stopPropagation()`

This is similar to event cancelling, but defers that cancellation until the end of the current handler. Any pending dispatches to the current `EventTarget` will be honored however.

Propagation of events to any parent node when bubbling is used or child nodes when event capturing is used will be inhibited.

Event.target (Property)

A property containing a reference to the object that the event was directed at.

Availability:	DOM level – 2 JavaScript – 1.2 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`EventTarget` object	
JavaScript syntax:	N	*myEvent*`.target`

You can construct an event handler that is shared amongst several target objects in Netscape. If that is the case, this property allows you to determine which one the event was aimed at, so that you can modify the event handler behavior accordingly.

DOM level 2 introduces the `EventTarget` object as a standard property, so it should be expected to be available more widely in due course than in the Netscape browser.

See also:	`Event.srcElement`, `MutationEvent.initMutationEvent()`

Property attributes:

`ReadOnly`.

Event.timeStamp (Property)

A time value at which the event was triggered.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myEvent*`.timeStamp`

This indicates the exact time (in milliseconds) when the event occurred. This should be equivalent to having called `Date.getUTCMilliseconds()` to obtain a measurement of the number of milliseconds since midnight on the first of January 1970.

The example demonstrates how this might be used to display an `alert()` dialog that indicates the number of milliseconds that have elapsed since the page was loaded.

The time stamp value here is the time the event was triggered. This is important because if you measure the time during an event handler, that may be some time after the event occurred. It may be important to collect a series of events and arrange them into sequence. The trigger time allows you to do this.

Warnings:

- The standard expects that this may not always be available, and that in such cases a value of zero should be returned.
- This feature appears to manifest a bug and unfortunately, the `Event` object does not seem to work with this property in the initial release of Netscape 6.0. This event model is a completely new implementation based on the DOM level 2 standard and so we might expect some instabilities in the early days.

Example code:

```
<HTML>
<HEAD>
<SCRIPT>
myLoadTime = new Date().getTime();
</SCRIPT>
</HEAD>
<BODY>
```

```
<SCRIPT>
document.write(myLoadTime);
</SCRIPT>
<HR>
<INPUT TYPE="BUTTON" VALUE="TOM" ONCLICK="measureTime(event);">
<SCRIPT>
function measureTime(anEvent)
{
   alert(anEvent.timeStamp- myLoadTime);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Date` object, `Date.getUTCMilliseconds()`

Property attributes:

`ReadOnly`.

Event.toElement (Property)

The object to which the mouse is moving.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Object` object	
JavaScript syntax:	IE	*`myEvent`*`.toElement`

This property is useful in `MouseOut` event handlers. It is a reference to the object that the mouse is just entering when it left an object. That object receives a `MouseOver` event. It also knows about the `MouseOut` object via the `fromElement` property of the `Event` object.

See also:	`Event.fromElement`

Property attributes:

`ReadOnly`.

Event.type (Property)

A string that contains the event type name.

Availability:	DOM level – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myEvent*.type

The event types in pre-version 6.0 examples of Netscape are determined by matching against a bitmask, which is then used internally for capturing and routing events. Eventually during an event handler, an event is passed to a handler and the `type` property is set to a string that describes the event type.

The DOM level 2 event handling that is supported by Netscape version 6 works more like it did for the MSIE browser, although the event bubbling and event capture techniques are both available. The event type property is still available although the DOM level 2 event specification does not list a set of values but does mention some possible values incidentally. It does indicate that the value should be an XML name value as defined in the XML specification which simply describes its 'well formedness' but does not enumerate any valid event type values.

Certain name values are reserved. For example, the XML standard reserves all names beginning with "XML". The value is case-insensitive and this rule should apply after conversion from international characters to locale specific characters.

Likewise, the DOM standard reserves all names beginning with the string "DOM". It is also likewise case- and internationalization-insensitive in its reservation of this name start string.

The specification suggests that third party event definitions that are implementation-specific should include some kind of prefix to avoid namespace collisions.

Here are some Event type names that can be observed by closely inspecting the DOM level 2 event specification:

Value	Event object type
HTMLEvents	`HTML Element` objects
MouseEvents	`MouseEvent` object
MutationEvents	`MutationEvent` object
UIEvents	`UIEvent` object

The standard notes that Key events are not yet supported and will be added in a later DOM level.

See also:	`DocumentEvent.createEvent()`, Event type constants, `Event.initEvent()`, XML name

Property attributes:

`ReadOnly`.

Web-references:

http://www.w3.org/TR/1998/Rec-xml-19980210

Event.which (Property)

The number of the mouse button or the code point value of a key that was pressed in Netscape.

Availability:	JavaScript – 1.2 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myEvent*`.which`

Netscape uses a completely different means of determining which mouse button or keyboard key was pressed. It also detects *[alt]*, *[control]* and *[shift]* modifier keys using a different technique to that used by MSIE.

The mouse buttons don't even correspond to the same mapping as MSIE. Here are the mouse button value passed in the `which` property:

- 1 – The left button
- 2 – The right button in a two button mouse
- 2 – The middle button in a three button mouse
- 3 – The right button in a three button mouse

If the event is a keyboard triggered event, the `which` property will contain the ASCII character value of the keyboard key that was pressed. Note that it is the ASCII value and not the Unicode value although the distinction is very subtle.

DOM level 2 event handling replaces this property of the `Event` object with the `MouseEvent.button` property.

Warnings:

- This is not supported by MSIE, which uses the `button` and `keyCode` properties to return mouse button values and key codes separately from one another.

See also:	`Event.button`, `Event.keyCode`, Keyboard events, `MouseEvent.button`, `onClick`, `onDblClick`, `onKeyDown`, `onMouseDown`, `onMouseUp`

Property attributes:

`ReadOnly`.

Event.width (Property)

The new width of a resized window or frame.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`MyEvent`*`.width`

If a window or frame is resized by the user, the new height value is available in this property.

See also:	`Event.height`

Event.x (Property)

The X coordinate of the event within a positioned object.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myEvent`*`.x`

When this event is passed with a resize event in Netscape, this property indicates the width of the object having been resized.

Otherwise, in Netscape, it indicates the horizontal position of the mouse pointer measured relative to the layer in which the event occurred.

In MSIE, it indicates the horizontal position of the mouse pointer, measured relative to the parent element.

Property attributes:

`ReadOnly`.

Event.y (Property)

The Y coordinate of the event within a positioned object.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myEvent`*`.y`

When this event is passed with a resize event in Netscape, this property indicates the height of the object having been resized.

Otherwise, in Netscape, it indicates the vertical position of the mouse pointer measured relative to the layer in which the event occurred.

In MSIE, it indicates the vertical position of the mouse pointer, measured relative to the parent element.

Property attributes:

`ReadOnly`.

Event propagation (Definition)

The means whereby events are passed up a hierarchy of objects.

Event propagation is important and handled slightly differently in Netscape versus MSIE.

Events are processed by handlers in objects that support the event and are then passed upwards, being handled as necessary as they traverse a tree. This is especially useful when a single `event` object represents the event, as properties can be added or modified at each stage.

Propagation also saves you proliferating event handlers all over your script. It may be that you can handle all the errors you need to by attaching a handler at a high level and then waiting for events to be propagated up to it.

This would be fine apart from the fact that Netscape and MSIE model this in quite different ways. The only common factor between them is that events ultimately end up at the `window` or `document`. They arrive there by quite different means, but at least you do get one centralized opportunity to handle them.

Netscape prior to version 6.0 does it like this. First it makes a list of events that it wishes to capture. This is done with the `captureEvents()` method. Once the event capture bitmask is set up, you need to register some event handlers by assigning a reference to a function object to the event handler properties. Events are then only propagated by the event handler if it chooses to do so. The `routeEvent()` function passes the event to the next event that has signified an interest with the `captureEvents()` bitmask. Alternatively, you can use the `handleEvent()` function to pass the event to an object which will then select an appropriate handler. Event propagation in Netscape is very flexible but needs to be set up manually very carefully. Basically, events are captured at the top and passed down the tree to the leaf nodes.

MSIE implements an event propagation technique that operates in completely the opposite direction. Instead of starting the processing of events at the top of the DOM hierarchy, events start at the bottom and work their way upwards. This is more of a traditional event propagation model. If an event handler wants to inhibit the passing upwards of an event, it can do so by setting the `cancelBubble` property of the `Event` object to true. The so called Input or Raw events exhibit this bubbling effect while the Semantic events do not.

At version 6.0 of Netscape, the DOM level 2 event model is supported. This a complete reorganization of event modelling in Netscape and fits more closely with the event model in MSIE. The DOM standard appears to reach a helpful compromise though and allows for both kinds of event propagation. This event model is still undergoing some change and there are additional features at DOM level 3 which have yet to be ratified and are not implemented in any browser yet.

Warnings:

- Not all kinds of events in MSIE bubble up through the DOM hierarchy. This is because there are two distinct types of events. These are sometimes called raw events and semantic events. Raw events bubble and semantic events do not.

See also:	`CaptureEvents()`, `Document.captureEvents()`, `Document.releaseEvents()`, `Element.onevent`, Event bubbling, `Event.cancelBubble`, `Layer.captureEvents()`, `Layer.releaseEvents()`, Semantic event, `Window.captureEvents()`, `Window.handleEvent()`, `Window.releaseEvents()`, `Window.routeEvent()`

Event type constants (Constant/static)

Netscape defines a set of constants that represent event types.

Netscape provides some static properties of the `Event` object class that define mask values for event types. These can then be combined with a bitwise OR operator to build a mask that can be applied with the `captureEvents()` and `releaseEvents()` methods that are supported by `Window`, `Document` and `Layer` objects.

Here is a list of the available Event type constants:

Bit	Constant	Event type
2^0	MOUSEDOWN	The mouse button was pressed while the pointer was over the element.
2^1	MOUSEUP	The mouse button was released while the pointer was within the extent region of the element.
2^2	MOUSEOVER	The mouse has just moved over the extent region of the element.
2^3	MOUSEOUT	The mouse has just moved out of the extent region of the element.
2^4	MOUSEMOVE	The mouse was moved within the extent region of the element.
2^5	undefined	Reserved for future use.
2^6	CLICK	The element has been clicked on.

Table continued on following page

Bit	Constant	Event type
2^7	DBLCLICK	The element has been double-clicked on.
2^8	KEYDOWN	A key was pressed while the element had focus.
2^9	KEYUP	A key was released while the element had focus.
2^10	KEYPRESS	A key was pressed and released again while the element had focus.
2^11	DRAGDROP	Some entity has been dragged over and dropped onto the element.
2^12	FOCUS	The element has had input focus restored to it.
2^13	BLUR	Window or input element has lost input focus.
2^14	SELECT	An item in a pop-up menu was selected.
2^15	CHANGE	The input element's value has changed.
2^16	RESET	The [RESET] button within a <FORM> was clicked.
2^17	SUBMIT	The [SUBMIT] button within a <FORM> was clicked.
2^18	undefined	Reserved for future use.
2^19	LOAD	An element (usually a <BODY> or <IMG>) has completed loading.
2^20	UNLOAD	The element (usually a <BODY>) is about to be unloaded.
2^21	undefined	Reserved for future use.
2^22	ABORT	Image loading was aborted.
2^23	ERROR	A script error has occurred.
2^24	undefined	Reserved for future use.
2^25	MOVE	The element (usually a Window) was moved.
2^26	RESIZE	The element (usually a Window) was resized.
2^27	undefined	Reserved for future use.
2^28	undefined	Reserved for future use.
2^29	undefined	Reserved for future use.
2^30	undefined	Reserved for future use.
2^31	undefined	Reserved for future use.

See also:	`captureEvents()`, `Document.captureEvents()`, `Document.releaseEvents()`, `Event` object, `Event.eventPhase`, `Event.type`, Keyboard events, `Layer.captureEvents()`, `Layer.releaseEvents()`, `Window.captureEvents()`, `Window.releaseEvents()`

EventCapturer object (Object/Navigator)

A special object that Netscape uses to grab events.

Availability:	JavaScript – 1.2 Netscape – 4.0	
JavaScript syntax:	N	*myEventCapturer* = window.open()

An object of this class is created when a `window.open()` method is executed in Netscape. Within the window, the `window` object is visible to the scripts running inside it. From outside, the creating script is handed an `EventCapturer` object as a handle on the window. In MSIE, you get a genuine `window` object back.

The properties of the Netscape `EventCapturer` object are difficult to inspect because they cannot be enumerated directly. The object, like many internal objects in Netscape has a constructor which can be inspected but the object is opaque otherwise.

Because the event manager complex is so new (and somewhat fragile in Netscape 6.0) it is hard to determine whether this is still true in that version. Some work needs to be done and a revised version of Netscape will be required before this can be verified.

See also:	`Window` object, `Window.open()`

EventException object (Object/DOM)

An object that describes the kind of event based exception that has occurred.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myEventException* = new EventException()
Object properties:	`code`	
Class constants:	`UNSPECIFIED_EVENT_TYPE_ERR`	

During the processing of an event with its handler, some unexpected circumstance may cause an exception to occur. This is not quite the same as a `try ... catch` exception as defined in ECMAScript.

DOM level 2 describes an `EventException` object, but it's not obvious how this would be associated with any JavaScript function that could be set up to catch it. The DOM level 2 event specification is founded on the principle that the implementation should be at least ECMAScript (third edition) compliant. This should ensure it has sophisticated enough error handling support to cope with an exception being thrown. Neither specification describes in detail exactly how this should happen and so it is likely that, whilst we may have browsers that support a standard and consistent object model, we still have portability problems because they implement the surrounding infrastructure differently.

Assuming that a function handler were called, it would be passed a reference to an `EventException` object from which it could obtain a code value that describes the cause of the exception. Perhaps, the `onError` handler may be used to trap this but an implementation might choose to provide an `onException` handler.

At DOM level 2, only the `UNSPECIFIED_EVENT_TYPE_ERR` is defined as a static constant. This has the value 0. Other event types may be defined in later versions of the DOM specification.

See also:	EventTarget.dispatchEvent()

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
code	1.5 +	-	6.0 +	-	-	2 +	-

EventException.code (Property)

A property containing a code value that indicates which one of the available set of exceptions has occurred.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	N	*myEventException*.code

At DOM level 2, the event model only supports a single UNSPECIFIED_EVENT_TYPE_ERR code value.

It is expected that other exception values will be defined in subsequent levels of the DOM standard as they are released.

EventListener object (Object/DOM)

A script function that is called when an event is triggered.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myEventListener* = function Listener(*anEvent*) { ... }
Argument list:	*anEvent*	A placeholder argument

This is an event handler function which you can define in the script source text and can then register as the listener for an event by means of the addEventListener() and removeEventListener() methods belonging to the EventTarget object.

The script function takes a single argument which is an Event object that is instantiated as the event is triggered.

Warnings:

- If you copy a document node with the `cloneNode()` method, the copies will not inherit the same listeners and you will need to add new listeners to handle any events that are dispatched to the new copies of the nodes.
- HTML 4.0 describes a way to associate listeners with objects as they are instantiated by means of the HTML tag attributes. However, to support multiple listeners, the internal mechanisms no longer use the member attribute mechanism for associating listeners with targets. This means that code that assigns values to the `onEventHandler` property family may no longer work as expected if you mix old and new style event handling techniques. Newer versions of browsers may deprecate the old way and may in subsequent versions render it unsupported.
- The DOM level 2 event model is somewhat ambiguous about this part of the event handling complex. It describes an event listener as having a `handleEvent()` method which is called. It is not clear how you would add a `handleEvent()` method as a member of a function object which was itself the element that was registered to receive the event.
- There may be changes to this aspect of the event model in later DOM levels to better support HTML based event specification.

See also:	`EventTarget.addEventListener()`, `MutationEvent` object

EventTarget object (Object/DOM)

A set of properties and methods that extend the behavior of DOM nodes to support event handling.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myEventTarget* = *myEvent*.currentTarget
	N	*myEventTarget* = *myEvent*.target
	N	*myEventTarget* = *myMouseEvent*.currentTarget
	N	*myEventTarget* = *myMouseEvent*.relatedTarget
	N	*myEventTarget* = *myMouseEvent*.target
	N	*myEventTarget* = *myMutationEvent*.currentTarget
	N	*myEventTarget* = *myMutationEvent*.target
	N	*myEventTarget* = *myUIEvent*.currentTarget
	N	*myEventTarget* = *myUIEvent*.target
Object methods:	`addEventListener()`, `dispatchEvent()`, `removeEventListener()`	

Netscape version 6 introduces this capability and adds the `EventTarget` methods and properties to its `Node` objects. This provides the necessary tools for registering and deregistering event listeners and for dispatching an event to a `Node`.

See also:	`MouseEvent.initMouseEvent()`, `MouseEvent.relatedTarget`, `MutationEvent.initMutationEvent()`, `Node` object

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
addEventListener()	1.5 +	-	6.0 +	-	-	2 +	-
dispatchEvent()	1.5 +	-	6.0 +	-	-	2 +	-
removeEventListener()	1.5 +	-	6.0 +	-	-	2 +	-

EventTarget.addEventListener() (Method)

Add a listener function to handle events dispatched to the owning `EventTarget` node.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myEventTarget*.addEventListener(*aType*, *aListener*, *aFlag*)
Argument list:	*aType*	A string containing the event type
	aListener	A reference to an `EventListener` function object
	aFlag	A Boolean value containing the `useCapture` disposition

The `addEventListener()` method will attach a function object to an `EventTarget` node in such a way that the function is called when the event is triggered. An `Event` object is passed to the function as an argument so you need to specify this in its declaration. This is how handler functions are registered as listeners.

You can replace a listener with another by registering a new listener with the same parameters. The old one will be discarded. If you want to stop the listener from responding to the event, you can deregister it with the `removeEventListener()` method.

The three parameters to the `addEventListener()` method are:

- A string describing the event type being registered.
- A reference to a function object that has been declared elsewhere in the script source text.
- A Boolean flag value that indicates whether the event listener should use bubbling (bottom up) or capture (top down) event propagation. Both types can be registered separately but must also be removed separately.

Examining the DOM level 2 specification suggests that these event type strings may be used:

Event string	Event object type	DOM
abort	HTMLEvent	2
blur	HTMLEvent	2
change	HTMLEvent	2
click	MouseEvent	2
DOMActivate	UIEvent	2

Table continued on following page

Event string	Event object type	DOM
DOMAttrModified	MutationEvent	2
DOMCharacterDataModified	MutationEvent	2
DOMFocusIn	UIEvent	2
DOMFocusOut	UIEvent	2
DOMNodeInserted	MutationEvent	2
DOMNodeInsertedIntoDocument	MutationEvent	2
DOMNodeRemoved	MutationEvent	2
DOMNodeRemovedFromDocument	MutationEvent	2
DOMSubtreModified	MutationEvent	2
error	HTMLEvent	2
focus	HTMLEvent	2
load	HTMLEvent	2
mousedown	MouseEvent	2
mousemove	MouseEvent	2
mouseout	MouseEvent	2
mouseover	MouseEvent	2
mouseup	MouseEvent	2
reset	HTMLEvent	2
resize	HTMLEvent	2
scroll	HTMLEvent	2
select	HTMLEvent	2
submit	HTMLEvent	2
unload	HTMLEvent	2

Note that the DOM level 2 event module specification is somewhat ambiguous on the specifics of event type values and you may want to refer to the specification if you are intending to exercise these capabilities in the Netscape Navigator version 6.0 browser. This functionality is fairly new and still in a state of evolution but since it is defined in a standard, it will become more widespread and is therefore the best way to ensure portability even if it takes a while for the various browser manufacturers to catch up and make their products compliant.

See also:	`Element.onevent`, `EventListener` object, `EventTarget.removeEventListener()`, `MutationEvent` object

EventTarget.dispatchEvent() (Method)

Create and send an event trigger to a target node.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myEventTarget*`.dispatchEvent(`*anEvent*`)`
Argument list:	*anEvent*	A reference to an `Event` object

Events can be created by your script independently of any other kind of event triggering. Indeed, they can be triggered using this mechanism from within the event handling complex because the event mechanisms specified by DOM level 2 must be re-entrant. That means they may be called recursively without any cross coupling of the local storage between nested or concurrent handlers.

Events triggered by this mechanism are handled in the same way as implementation generated events.

You must first create an `Event` object to pass as an argument to this method when you call it. The resulting Boolean value tells you whether the `preventDefault` flag was set by any of the listeners that were invoked. If this value is true, then any default behavior should be suppressed. It is quite important to provide a consistent level of support for this so that it behaves in the same way as an implementation controlled event dispatcher.

During the handling of the event being dispatched by this method, an `EventException` may be raised.

At DOM level 2, only the unspecified event type is available as an exception. This will be raised by an `unspecified` or `null` event type value.

See also:	`EventException` object

EventTarget.removeEventListener() (Method)

Deregister an event listener that was established with the `addEventListener()` method.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myEventTarget*`.addEventListener(`*aType*`,` *aListener*`,` *aFlag*`)`
Argument list:	*aType*	A string containing the event type
	aListener	A reference to an `EventListener` function object
	aFlag	A Boolean value containing the `useCapture` disposition

The `removeEventListener()` method will detach a previously registered function object from an `EventTarget` node. This will stop the event being handled by that target.

The three parameters to the `removeEventListener()` method are:

- A string describing the event type being registered.
- A reference to a function object that has been declared elsewhere in the script source text.
- A Boolean flag value that indicates whether the event listener should use bubbling (bottom up) or capture (top down) event propagation. Both types can be registered separately but must also be removed separately.

See also:	`Element.onevent`, `EventTarget.addEventListener()`

Exactly equal to (===) (Operator/identity)

The two values must be identically equal in value and type.

Availability:	ECMAScript edition – 3

Refer to:

Identically equal to (`===`)

Cross-references:

ECMA 262 edition 3 – section – 11.9.4

Exception (Definition)

An unexpected result from an expression evaluation.

An exception occurs when an expression yields a result that was not expected when you evaluate it. In numerical expressions, the `NaN` value is provided for just such a circumstance. `NaN` represents a numeric quantity that cannot be resolved within the range of meaningful values. The interpreter knows it is numeric but the value is wrong. The `undefined` and `Infinity` values also help in the management of such exceptions.

In general, the exception handling in JavaScript is better than that in a compiled language and therefore it is more forgiving. Its best efforts will usually be good enough to yield a working script where a compiled program may fail with a fatal error.

The DOM level 1 specification describes an enumerated set of exception codes as follows:

Value	Name	DOM
1	INDEX_SIZE_ERR	1
2	DOMSTRING_SIZE_ERR	1
3	HIERARCHY_REQUEST_ERR	1
4	WRONG_DOCUMENT_ERR	1
5	INVALID_CHARACTER_ERR	1
6	NO_DATA_ALLOWED_ERR	1
7	NO_MODIFICATION_ALLOWED_ERR	1
8	NOT_FOUND_ERR	1
9	NOT_SUPPORTED_ERR	1
10	INUSE_ATTRIBUTE_ERR	1
11	INVALID_STATE_ERR	2
12	SYNTAX_ERR	2
13	INVALID_MODIFICATION_ERR	2
14	NAMESPACE_ERR	2
15	INVALID_ACCESS_ERR	2

See also:	Expression, Infinity, `NaN`, undefined

Exception handling (Definition)

The process of managing errors.

This is enhanced in JavaScript version 1.3 by the addition of the `try ... catch` and `throw` statements.

See also:	`catch( ... ),throw,try ... catch ... finally`

execScript() (Method)

Execute a script.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`User defined`	
JavaScript syntax:	IE	`execScript(`*`aSourceText`*`)`
	IE	*`myWindow`*`.execScript(`*`aSourceText`*`)`
Argument list:	*`aSourceText`*	Some legal JavaScript source

Refer to:

`Window.execScript()`

Executable code (Definition)

Script code that can be parsed and run as a program or function.

Availability:	ECMAScript edition – 2

Executable code is from the programmer's point of view a block of script source text that is parsed and executed by the interpreter.

There are five distinct types of executable code:

- Global code
- Eval code
- Function code
- Anonymous code
- Implementation-supplied code

See also:	Anonymous code, Eval code, Execution context, Function code, Global code, Implementation-supplied code

Cross-references:

ECMA 262 edition 2 – section – 10.1.2

ECMA 262 edition 3 – section – 10.1.2

Execute a function (Definition)

The act of calling a function during script execution.

Refer to:

Function call

Execution context (Definition)

An environment in which the script source code is executed.

Availability:	ECMAScript edition – 2

An execution context is the environment within which a portion of script code executes. If subsequent fragments of code are called, they are assembled to form a stack of execution contexts such that when one exits, the stack is popped and control returns to the immediate parent or caller.

The first item to be placed on the stack is the initial code that is considered to be global and is executed in a web browser as the page is parsed. Other types of host implementation may execute this global code at a different time. The topmost item on the stack is the code fragment that is currently executing. This might be the code in a called function or some code that is triggered by an event and is running as the handler for it.

Every execution context has associated with it its own variable object in which functions, formal parameters and variables are maintained as properties.

Each execution context also has its own private scope chain, which is a logical list of objects which are searched when name binding an identifier.

As the flow of control enters each execution context, its scope chain is created and initialized, variable instantiation is performed creating the variable object which in turn causes the arguments object to be created as well. At this time the value is also determined The particular values for these items depend on the kind of code being entered.

See also:	`Activation` object, Anonymous code, `argc` parameter, `Arguments` object, `argv` parameter, Eval code, Executable code, Function code, `Function` object, `Function.arguments[]`, Global code, `Global` object, Identifier resolution, Implementation-supplied code, `main()` function, Primary expression, Scope chain, `this`, Variable instantiation

Cross-references:

ECMA 262 edition 2 – section – 10

ECMA 262 edition 3 – section – 10

Execution environment (Definition)

The environment in which a script is executed.

The execution environment is where the script is run. In compiled languages, the translation environment and the execution environment may not be the same. JavaScript is interpreted and therefore the translation and execution environments are one and the same.

However, JavaScript can be transported across a network and may be executed in a server and then transferred to a client and executed there.

This means that the execution environment may change from time to time. The script should be aware of the context and environment it is being executed in if it needs to be portable in this way.

Future developments suggest that JavaScript may be available in hosted environments that will require the script to be delivered in a compact form. Indeed, some JavaScript interpreters are written in Java and decompose the script to Java byte-codes before executing them in the Java Virtual Machine. If these tokenized JavaScripts were transmitted to a remote machine, then JavaScript could be considered to be a compiled (or semi-compiled) language. In that case the translation environment would be different to the execution environment.

See also:	`argc` parameter, `argv` parameter, Environment, Garbage collection, Host environment, `main()` function, Script termination

Cross-references:

Wrox *Instant JavaScript* – page – 5

Exponent-log function (Definition)

Functions that calculate exponents and logs.

The ECMAScript standard defines several functions that calculate exponents and logs and several other value properties that are useful in this context:

- `Math.E`
- `Math.exp()`
- `Math.LN2`
- `Math.LN10`
- `Math.log()`
- `Math.LOG2E`
- `Math.LOG10E`

See also:	Integer-value-remainder, `Math` object, `Math.E`, `Math.exp()`, `Math.LN10`, `Math.LN2`, `Math.log()`, `Math.LOG10E`, `Math.LOG2E`, Power function, Trigonometric function

export (Statement)

Exports some properties to allow them to be imported into another execution context.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 Netscape – 4.0 Netscape Enterprise Server – 3.0	
JavaScript syntax:	N	`export` *`aFunction`*`;`
	N	`export` *`aProperty`*`;`
Argument list:	*`aFunction`*	A function object to export
	`aProperty`	a named property

ECMAScript edition 2 suggests this is a future extension. As of the third edition of the ECMAScript standard it is still denoted as a reserved word.

Netscape 4 anticipates that a future standard will endorse this capability and provides it anyway.

This functionality allows layers to define handlers for themselves and then export them to allow other layers or windows to call them.

This facility is also useful to allow controlled access via the security policy. This can then allow an unsigned script to have access to content in a signed script's context.

This is good Object Oriented Programming technique on the grounds that hiding the private data and making a public interface available means code can be reused. This black-box approach is much used in languages such as Java, SmallTalk and Objective-C.

Warnings:

- This only works in Netscape version 4 when the `LANGUAGE` attribute is set to "JavaScript1.2". This will affect the behavior of the `==` and `!=` operators as well.
- This can affect the security policy regarding the "same-signer" trustworthiness of a page.
- Be careful that you do not export a secure method or property and allow it to be executed or seen by insecure and untrusted non-signed scripts running in other windows.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
var myLocalVariable;
function myLocalFunction()
{
   document.write("Test");
}
export myLocalVariable;
export myLocalFunction;
</SCRIPT>
</BODY>
</HTML>
```

See also:	`import`, Same origin, Signed scripts

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Expression (Definition)

Combining one or more operands with an operator creates an expression.

Availability:	ECMAScript edition – 2

An expression consists of an operand on its own, or a combination of operators and operands. An operand may be a constant value, a functional value, or a variable. Evaluating expressions can easily cause side effects, especially when functions are invoked.

Expressions are the contexts within which JavaScript operates on entities in its object model. There are a variety of different kinds of expressions according to whether you are referring to the destination or source of an assignment or reference.

Expressions fall into one of several categories according to the operator used in the expression.

The following expression types are defined in the ECMAScript standard:

Type	Description
Primary	A Primary Expression is a specific object, identifier or literal and may also be the result of evaluating another nested expression when it is surrounded by the grouping operators (round brackets).
Left-Hand-Side	This kind of expression identified the destination of an assignment (even if that assignment operation is only implied).
Postfix	Postfix expressions operate on Left-Hand-Side (sometimes called LValue) expressions.
Unary	Unary expressions can also be considered to be prefix expressions and also operate on LValues.
Multiplicative	Multiplicative expressions use the multiplicative operators to yield a result by operating on two other values which may themselves be nested.
Additive	Additive expressions use the additive operators to yield a result by operating on two other values which may be nested expressions.
Shift	Shifts the left value by an amount specified by the right value.
Relational	Relational expressions yield a Boolean result according to the relational test of the values either size of the operator.
Bitwise	Bitwise expressions perform a bit by bit operation across the entire integer width of the values.
Logical	Logical operators perform a test of the Boolean value of the two operands either side of the operator.
Conditional	Conditional expressions test a logical expression and perform one of two possible alternative code blocks.
Assignment	Assignment expressions can be broken down into a two-operand expression with the result being assigned to the value on the left.
Comma	Comma expressions occur rarely and are used to evaluate several expressions at once.

The operators are discussed in detail in individual topics. Refer to those topics for more details.

Expressions resolve eventually to a primary expression, which has a distinct value.

In a compound expression, the precedence (or "Who's on first") is governed by the operator selected for that sub-expression. However that operator-driven precedence model can be overridden by use of the grouping operators.

See also:	Associativity, Bit-field, Exception, JavaScript language, Side effect

Cross-references:

ECMA 262 edition 2 – section – 11

ECMA 262 edition 3 – section – 11

Expression statement (Definition)

Expressions do not necessarily have to be placed on the right of an assignment operator.

Availability:	ECMAScript edition – 2

An expression statement is an expression that is evaluated on a line by itself.

Expression statements consist of a stand-alone expression followed by a semi-colon (;).

An example of the main use of expressions in this context would be a function call. Since a function is a named block of code that can be called by putting its identifier into an expression, a function that does not return a value or where the value has been voided is being called like a procedure or sub-routine in other languages.

See also:	Constant expression, Semi-colon (;), Statement

Cross-references:

ECMA 262 edition 2 – section – 12.4

ECMA 262 edition 3 – section – 12.4

Wrox Instant JavaScript – page – 18

extends (Reserved word)

Reserved for future language enhancements to do with better class handling.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

external (Property)

Reference to external objects outside of the interpreter.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`External` object	
JavaScript syntax:	IE	`external`
	IE	*`myWindow`*`.external`

See also:	`Window.external`

external object (Object/JScript)

Since MSIE can be embedded as a component into other applications, this object represents the object model of such a containing application.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*`myExternal`* `= external`
	IE	*`myExternal`* `=` *`myWindow`*`.external`
Object properties:	`menuArguments`	
Object methods:	`AddChannel(),AddDesktopComponent(),AddFavorite(), AddFavourite(),AutoCompleteSaveForm(),AutoScan(), ImportExportFavorites(),ImportExportFavourites(), IsSubscribed(),NavigateAndFind(),ShowBrowserUI()`	

Part of the MSIE on Windows environment object space. You should avoid using this facility if you want your scripts to be usable outside of the Windows environment.

To use this effectively, you need to know quite a bit about the application whose components you are calling in using this functionality. Each application has its own object model and unless you know how it works, making use of it will be a bit 'Hit and Miss'.

Warnings:

- Several methods for this object have arguments that are mandatory but may take empty strings or `null` values. This is somewhat diametrically opposed to the normal JavaScript behavior which dictates that unnecessary arguments are optional and can be assumed to be a default value when they are not present.

- By implementing methods with mandatory `null` or empty string values, Microsoft puts the script developer off balance and can cause unnecessary difficulty when developing scripts. Novices and even expert scripters may be caught out by this kind of implementation.
- You should always check the reference material if you are in any doubt as to how an API interface to a method is designed to work.

Property	JavaScript	JScript	N	IE	Opera	Notes
`menuArguments`	-	5.0 +	-	5.0 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`AddChannel()`	-	5.0 +	-	5.0 +	-	-
`AddDesktopComponent()`	-	5.0 +	-	5.0 +	-	-
`AddFavorite()`	-	5.0 +	-	5.0 +	-	-
`AddFavourite()`	-	3.0 +	-	4.0 +	-	Warning
`AutoCompleteSaveForm()`	-	5.0 +	-	5.0 +	-	-
`AutoScan()`	-	5.0 +	-	5.0 +	-	-
`ImportExportFavorites()`	-	5.0 +	-	5.0 +	-	Warning
`ImportExportFavourites()`	-	3.0 +	-	4.0 +	-	Warning
`IsSubscribed()`	-	5.0 +	-	5.0 +	-	-
`NavigateAndFind()`	-	5.0 +	-	5.0 +	-	-
`ShowBrowserUI()`	-	5.0 +	-	5.0 +	-	Warning

external.AddChannel() (Method)

A means of adding a channel to an `external` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myExternal*.AddChannel(*aURL*)
Argument list:	*aURL*	The URL for a CDF file

With this method, you can use the Microsoft Active Channel system to share content delivered via channels.

This presents a dialog box allowing you to select a channel to add the channel defined in the argument or to specify a new one.

The argument value passed is a URL that points at a valid CDF file.

If the operation fails, an error event is generated and unless you are trapping the `onError` event with your own handler, the MSIE browser will present an error alert box.

See also:	`external.IsSubscribed()`, File extensions

external.AddDesktopComponent() (Method)

A means of adding an image or web site link to the desktop workspace.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myExternal*.AddDesktopComponent(*aURL*, *aType*)
	IE	*myExternal*.AddDesktopComponent(*aURL*, *aType*, *aLeft*)
	IE	*myExternal*.AddDesktopComponent(*aURL*, *aType*, *aLeft*, *aTop*)
	IE	*myExternal*.AddDesktopComponent(*aURL*, *aType*, *aLeft*, *aTop*, *aWidth*)
	IE	*myExternal*.AddDesktopComponent(*aURL*, *aType*, *aLeft*, *aTop*, *aWidth*, *aHeight*)
Argument list:	*aURL*	The URL where the resource can be located
	aType	What kind of item is added to the desktop
	aLeft	Left edge position
	aTop	Top edge position
	aWidth	Size width
	aHeight	Size height

This provides a way of customizing the underlying desktop with shortcuts and images.

You must always specify the URL value and the type indicator. The remaining option arguments describe the position and size of the item being placed on the desktop.

The type value must be one of:

- image
- website

This only works if you have the Active Desktop support installed. If it is not installed, this method does nothing.

Example code:

```
//Add the Wrox web site as a desktop item
window.external.AddDesktopComponent("http://www.wrox.com", "website", 100, 100,
200, 200);
```

external.AddFavorite() (Method)

Support for external applications and objects when using MSIE in the Windows environment. This adds an item to the favorites collection.

Availability:	JScript – 5.0 Internet Explorer – 5.0

JavaScript syntax:	IE	*myExternal*.AddFavourite(*aURL*)
	IE	*myExternal*.AddFavourite(*aURL*, *aName*)
Argument list:	*aURL*	A Url for a link in the bookmarks list
	aName	The name that goes in the menu list

With this method, you can add your own bookmarks to the favorites menu.

You must specify a URL value (otherwise there is no point in adding the bookmark). However, you need not specify the second argument which is the text displayed in the menu.

Typically you might provide a button on the page called "bookmark me" which might call some script to work out what the real URL should be. This gets round one of the major problems with framed sites in that a bookmark usually yields unpredictable results since you are accessing the site at a sub-framed level.

Example code:

```
// Bookmark a page from its current location
// using its document title string as an
// identifying text.
window.external.AddFavorite(location.href, document.title);
```

external.AutoCompleteSaveForm() (Method)

Support for external applications and objects when using MSIE in the Windows environment. This accesses the auto-complete mechanisms.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myExternal*.AutoCompleteSaveForm(*aForm*)
Argument list:	*aForm*	A reference to a `form` object

This method provides a way to save the contents of the text and password fields in the auto-complete storage area of your computer. Normally this would happen when the form is submitted but you can call this method to save the values without submitting the form.

If necessary, you can override the auto-complete capabilities for individual fields by means of the `AUTOCOMPLETE` HTML tag attribute being applied to the appropriate form input elements.

This method takes one argument that is mandatory. This argument is an object reference to the form whose fields are to be stored in the auto-complete registers.

external.AutoScan() (Method)

Connects to a web server using a template and registry scanning process to work out the target web server address from a partial domain name.

Availability:	JScript – 5.0 Internet Explorer – 5.0

JavaScript syntax:	IE	`myExternal.AutoScan(aDomain, anErrURL)`
	IE	`myExternal.AutoScan(aDomain, anErrURL, aTarget)`
Argument list:	`aDomain`	The domain address of a web site
	`anErrURL`	A URL to display if `aDomain` is inaccessible
	`aTarget`	The target window or frame to put the content into

This mechanism is used to find a web site when you are not certain of the URL.

The Registry in the client system contains a list of top level domains such as .com, .net, .org and .edu which will be tried one at a time until a valid connection is completed.

If none of these turns out to yield a working site address, then the error URL is accessed and displayed instead.

The browser will add the leading "www." to the domain address so you might only need to specify a domain value of "wrox" for the browser to eventually connect to "www.wrox.com".

external.ImportExportFavorites() (Method)

A mechanism for storing favorites lists on a server and getting them back later.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	`myExternal.ImportExportFavourites(aFlag, aURL)`
Argument list:	`aFlag`	A direction indicator
	`aURL`	A storage URL location

This is useful in a corporate environment where you may want to define a set of standard favorites and download them to the user's browser when they access the corporate Intranet.

Both arguments are mandatory so you must specify the direction and server location. This also requires that you have a server set up correctly to handle these requests.

The flag value should be `false` to save the favorites list and `true` to restore from the server.

When the method is called, the browser will ask you to confirm that you want to upload or download the favorites collection. The server needs to be carefully set up to erase any favorites that were previously stored unless you want to gradually accumulate a larger and larger collection of stored favorites. This deletion on the server does not propagate into the browser and any favorites retrieved from the server are merged with the existing set in the client browser.

It is not clear from the documentation whether duplicates are eliminated and if they are not, then it is likely that a later version of the browser would provide this capability.

If you do not specify a location value, then the browser will open a file dialog so you can import and export to the local file system. You must still provide an argument, but it can be an empty string to trigger this behavior.

Warnings:

- Oddly, this method can both save and retrieve the favorites using a flag value to determine the direction. This is slightly unusual and many similar actions in JavaScript provide one method to save and another to restore. You need to be constantly on the lookout for these kinds of inconsistent behaviors when developing JavaScript applications.

external.IsSubscribed() (Method)

Channel handling support is provided by this enquiry that determines whether the channel is registered.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myExternal`*`.IsSubscribed(`*`aURL`*`)`
Argument list:	*`aURL`*	The URL for a CDF file

You might want to enhance the script that adds channels by checking first to see if the user has already subscribed to a channel. That way you can avoid asking them the same question every time they enter the page that gives them access to the channel.

This query method returns a Boolean value `true` if the client is already subscribed and `false` if not.

The argument would be the same URL value that is used in the `addChannel()` method.

See also:	`external.AddChannel()`

external.menuArguments (Property)

The `window` object where a context menu was executed.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Window` object	
JavaScript syntax:	IE	*`myExternal`*`.menuArguments`

Once you have this window, you can inspect its `event` object to determine various properties that tell you about the user interaction.

external.NavigateAndFind() (Method)

This augments the anchor linking behavior and adds a highlighting capability.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myExternal*.NavigateAndFind(*aURL*, *aString*, *aTarget*)
Argument list:	*aURL*	The URL to link to
	aString	A string to be selected and highlighted
	aTarget	The frame or window to load the content into

You can use this method to link to a page and then highlight some fragment of text when it is loaded.

Although the target frame argument is mandatory, it can be an empty string.

external.ShowBrowserUI() (Method)

Accesses dialog panels that are built into the browser.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myExternal*.ShowBrowserUI(*aName*, null)
Argument list:	*aName*	The name of a UI dialog

This method takes two arguments although one is obviously a place holder. Both arguments are mandatory. The first indicates the dialog to be displayed while the second is a `null` value.

The values for selecting browser UI dialogs are:

- `LanguageDialog`
- `OrganizeFavorites`

Warnings:

- The API for this method breaks a rule of requiring a place holder argument to be set to `null` and yet mandating that it must be present. This is poor API design. It is possible that the argument may take undocumented values other than `null` which may alter its behavior but it should still be optional.

Fade() (Filter/transition)

A transition effect with the appearance of a dissolve between two images.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

filter – `Fade()`

false (Primitive value)

The Boolean false value.

Availability:	ECMAScript edition –2
Property/method value type:	`Boolean primitive`

This is a Boolean primitive value representing the logically false state.

Conditional code execution depends on this value to signify the execution of a block of script code.

Warnings:

- Beware of a rather insidious effect when converting Boolean primitive values into objects. All objects yield the value `true` when converted back to a Boolean primitive. This also applies to a Boolean object having the value `false`.
- This `if()` test yields a `true` condition and selects the opposite branch to that which you would expect:

```
var myBoolean = new Boolean(false);

if(myBoolean)

{

 // This branch is called

}

else

{

 // You would have expected this one to be called

}
```

See also:	Boolean, Boolean, Boolean literal, Definition, `true`

Cross-references:

ECMA 262 edition 2 section 9.2

ECMA 262 edition 2 section 15.6

ECMA 262 edition 3 section 9.2

ECMA 262 edition 3 section 15.6

fdlibm (Product)

A publicly available library of math functions.

Availability:	ECMAScript edition –2

A freely distributable mathematical library available from Sun Microsystems. It supports the algorithms specified in IEEE 754. It is available by contacting fdlibm-comment@sunpro.eng.sun.com.

This library is commonly used to implement the mathematical functions belonging to the built-in native `Math` object.

Cross-references:

ECMA 262 edition 2 section 15.8.2

ECMA 262 edition 3 section 15.8.2

FIELDSET object (Object/HTML)

A field set within a form.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myFIELDSET* = *myDocument*.all.*anElementID*
	IE	*myFIELDSET* = *myDocument*.all.tags("FIELDSET")[*anIndex*]
	IE	*myFIELDSET* = *myDocument*.all[*aName*]
	-	*myFIELDSET* = *myDocument*.getElementById(*anElementID*)
	-	*myFIELDSET* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myFIELDSET* = *myDocument*.getElementsByTagName("FIELDSET")[*anIndex*]
HTML syntax:	`<FIELDSET> ... </FIELDSET>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`accessKey, align, form, margin, tabIndex`	
Event handlers:	`onAfterUpdate, onBeforeUpdate, onBlur, onChange, onClick, onDblClick, onDragStart, onErrorUpdate, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onScroll, onSelect, onSelectStart`	

A `FIELDSET` object represents a structured group of `Input` objects within a form. You can operate on them collectively. Grouping `Input` items might be useful for activating or hiding `Input` items without needing to access each one individually, which can be very useful if a form is undergoing rapid and continuous changes during its development. Controlling this with a `FIELDSET` allows the member elements to change without needing to modify your script.

See also:	`Element` object, `Input` object, `Input.accessKey`, `Legend` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
align	-	3.0 +	-	4.0 +	-	-	-	-
form	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
margin	-	3.0 +	-	4.0 +	-	-	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onChange	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-
onClick	1.5 +	3.0 +	6.0 +	3.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onResize	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onSelect	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

FIELDSET.align (Property)

The alignment of the field set within its surrounding objects.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myFIELDSET*.align

The alignment of the `FIELDSET` object with respect to its containing parent object is defined in this property. The following alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

FIELDSET.margin (Property)

The margin around a fieldset to separate it from its surrounding objects.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myFIELDSET*.margin

You may want to place a margin round the fieldset of `Input` objects on screen. This property contains the current width settings for that margin.

File extensions (Definition)

A means of determining file content by a part of its file name.

JavaScript can be contained in a variety of file types. The one that you use, depends on what you plan to do with the script and the kind of environment it is being used in. Here is a summary of the file types you are likely to encounter:

Extension	Description
.123	Lotus 123 Document
.3dmf	QuickDraw 3D File
.509	Certificates
.669	669 MOD Music
.8med	Amiga OctaMed music
.8svx	Amiga OctaMed music
.aam	Authorware
.aas	Authorware
.ai	Postscript Document
.aif	AIFF Audio
.aifc	AIFF Audio
.aiff	AIFF Audio
.al	Amiga OctaMed music
.ani	Animated NeoChrome
.apr	Lotus Approach Document
.arc	PC ARChive
.arj	ARJ Archive
.arr	Animated NeoChrome
.art	Animated NeoChrome
.au	AU Audio/ULAW Audio
.avi	Microsoft Video
.bar	Unix BAR Archive
.bga	OS/2 Bitmap
.bin	MacBinary File
.binary	Untyped Binary Data
.bmp	OS/2 Bitmap
.bmp	Windows Bitmap
.bw	OS/2 Bitmap
.c	Text File
.cdf	Channel definition file
.cer	Certificates
.cfg	Configuration file
.cgi	Common Gateway Interface dynamic page
.cgm	Animated NeoChrome
.ckl	Compromized Key List

Table continued on following page

Extension	Description
.class	Java Class File
.clp	Animated NeoChrome
.com	MS-DOS Executable
.cpio	Unix CPIO Archive
.cpt	Compact Pro Archive
.crl	Certificate Revocation List
.crt	Certificates
.csh	C Shell Program
.css	Text File
.ct	Animated NeoChrome
.cut	Animated NeoChrome
.cvs	Canvas Drawing
.dbf	DBase Document
.dcr	Shockwave/Director
.dcx	Animated NeoChrome
.dir	Shockwave/Director
.dl	DL Animation
.dll	MS-DOS Executable
.doc	Microsoft Word Document
.dot	Word for Windows Template
.dqy	Excel Worksheet
.dv	DV Video
.dvi	TeX DVI Document
.dxr	Director
.enc	Pre-encrypted Data
.eps	Encapsulated Postscript document
.epsf	Encapsulated Postscript document
.evy	Envoy Document
.exe	MS-DOS Executable
.fdf	Forms Data Format
.fif	Fractal Image Format
.fit	Flexible Image Transport
.flc	DV Video/FLC Animation
.fli	DV Video/FLC Animation
.fm	FileMaker Pro Database
.fm3	FileMaker Pro Database
.fts	Animated NeoChrome
.gem	Animated NeoChrome

Table continued on following page

Extension	Description
.gif	GIF Image
.gl	DL Animation
.grp	Animated NeoChrome
.gtar	GNU Tape Archive
.gz	GNU Zip Compressed Data
.h	Text File
.hcom	Amiga OctaMed music
.hdf	HDF Data File
.hpgl	Animated NeoChrome
.hqx	Macintosh BinHex Archive
.htc	Microsoft HTML Component
.htm	HTML web page
.html	HTML web page
.ic1	Animated NeoChrome
.ic2	Animated NeoChrome
.ic3	Animated NeoChrome
.icn	Animated NeoChrome
.ico	Animated NeoChrome
.ief	IEF image
.iff	Animated NeoChrome
.ilbm	Animated NeoChrome
.image	Apple DiskCopy Image
.img	Animated NeoChrome
.ini	Text File
.iqy	Excel Worksheet
.jar	Java archive file
.java	Java source file
.jfif	OS/2 Bitmap
.jfx	TIFF Image
.jpe	JPEG Image
.jpeg	JPEG Image
.jpg	JPEG Image
.js	JavaScript include file
.jsc	JavaScript configuration file
.kar	MIDI
.latex	LaTeX Document
.lbm	Animated NeoChrome
.lck	Old style configuration file

Table continued on following page

Extension	Description
.lha	LHArc Archive
.lwp	Lotus WordPro Document
.lzh	LHArc Archive
.m15	DV Video/MPEG video/audio stream
.m1a	MPEG audio stream
.m1a	MPEG video/audio stream
.m1s	DV Video/MPEG audio stream
.m1v	MPEG video/audio stream
.m2s	DV Video
.m2v	DV Video
.m3u	MP3 PlayLists
.m75	DV Video/MPEG video/audio stream
.mac	MacPaint Image/PICT Image
.mda	Microsoft Access Database
.mdb	Microsoft Access Database
.mde	Microsoft Access Database
.med	Amiga OctaMed music
.mid	MIDI Music
.midi	MIDI Music
.ml	ML Source
.mocha	JavaScript Program
.mod	Amiga OctaMed music
.moov	QuickTime Movie
.mov	QuickTime Movie
.mp2	MPEG video/audio stream
.mp2v	MPEG2 Video
.mp3	MPEG-1 Layer 3 audio stream/MPEG Movie
.mpa	MPEG video/audio stream
.mpe	MPEG video/audio stream
.mpeg	MPEG video/audio stream
.mpegv	MPEG Video
.mpg	MPEG video/audio stream
.mpm	MPEG video/audio stream
.mpv	MPEG video/audio stream
.mpv2	MPEG2 Video
.msp	Animated NeoChrome
.mtm	669 MOD Music
.mw	MacWrite Document

Table continued on following page

Extension	Description
.mwii	MacWrite Document
.neo	Animated NeoChrome
.nsc	application/x-conference
.nst	Amiga OctaMed music
.obj	MS-DOS Executable
.oda	ODA Document
.okt	Amiga OctaMed music
.or2	Lotus Organizer Document
.or3	Lotus Organizer Document
.org	Lotus Organizer Document
.otf	OpenType Font
.out	Untyped Binary Data
.ovl	MS-DOS Executable
.p7c	PKCS7 Encrypted Data
.p7m	PKCS7 Encrypted Data
.p7s	PKCS7 Signature
.pac	A proxy or parameter package file (also a NeoChrome image file)
.pbm	Portable Bitmap
.pc1	Animated NeoChrome
.pc2	Animated NeoChrome
.pc3	Animated NeoChrome
.pcd	PhotoCD Image
.pcs	Animated NeoChrome
.pct	PICT Image
.pcx	Animated NeoChrome
.pdf	Portable Document Format
.pf	Private File
.pgm	Portable Graymap
.pgp	PGP Key File
.pi1	Animated NeoChrome
.pi2	Animated NeoChrome
.pi3	Animated NeoChrome
.pic	PICT Image
.pict	PICT Image
.pit	PackIt Archive
.pkg	AppleLink Package
.pl	Perl Program
.pls	MP3 PlayLists

Table continued on following page

Extension	Description
.plt	Animated NeoChrome
.pm	Animated NeoChrome
.pm3	PageMaker 3 Document
.pm4	PageMaker 3 Document
.pm5	PageMaker 3 Document
.png	PNG Image
.pnm	PBM Image
.pnt	MacPaint Image
.pntg	OS/2 Bitmap/MacPaint Image
.pot	Microsoft PowerPoint Show
.ppa	Microsoft PowerPoint Show
.ppm	Portable Pixmap
.pps	Microsoft PowerPoint Show
.ppt	Microsoft PowerPoint Show
.pre	Lotus Freelance Document
.prz	Lotus Freelance Document
.ps	Postscript Document
.psd	PhotoShop Document
.pt4	PageMaker 3 Document
.pt5	PageMaker 3 Document
.pwz	Microsoft PowerPoint Show
.pxr	PhotoShop Document
.qcp	QCP Audio
.qdv	Animated NeoChrome
.qif	OS/2 Bitmap
.qt	QuickTime Movie
.qtc	video/x-qtc
.qti	QuickTime Image
.qtif	QuickTime Image
.qxd	QuarkXpress Document
.qxt	QuarkXpress Document
.ra	RealAudio Clip
.ram	RealPlayer File
.ras	CMU Raster Image
.raw	Animated NeoChrome
.rf	RealFlash Clip
.rgb	SGI Image/RGB Image
.rgba	SGI Image

Table continued on following page

Extension	Description
.rif	Animated NeoChrome
.rjs	RealSystem Skin
.rle	Animated NeoChrome
.rm	RealMedia File
.rmf	audio/rmf/audio/x-rmf
.rmm	RealPlayer File
.rmp	RealJukebox Music Package
.rmx	RealSystem Secure Media Clip
.rnx	RealPlayer File
.rp	RealPix Clip
.rpl	Replica Document
.rpm	RealPlayer Plugin
.rsc	Resource File
.rsml	RealSystem ML File
.rsrc	Resource File
.rt	RealText Clip
.rtf	Rich Text Format File
.rts	Real Time Streaming Protocol
.rtsp	Real Time Streaming Protocol
.rv	RealVideo Clip
.s3m	669 MOD Music
.sam	Lotus WordPro Document
.sc2	Microsoft Schedule+ Application
.scc	Animated NeoChrome
.scd	Microsoft Schedule+ Application
.scg	Animated NeoChrome
.sch	Microsoft Schedule+ Application
.sci	Animated NeoChrome
.scm	Lotus ScreenCam Movie
.scp	Animated NeoChrome
.scr	Animated NeoChrome
.scu	Animated NeoChrome
.sd2	DV Video
.sdp	Session Description Protocol/Scalable Multicast
.sea	Self-Extracting Archive
.sf	IRCAM Sound
.sgi	OS/2 Bitmap/SGI Image
.sgm	SGML Document

Table continued on following page

Extension	Description
.sgml	SGML Document
.sh	Bourne Shell Program
.shar	Unix Shell Archive
.shp	Animated NeoChrome
.shtm	HTML web page with server-side include
.shtml	HTML web page with server-side include
.sit	Macintosh StuffIt Archive
.sit	StuffIt Archive
.six	Animated NeoChrome
.smf	MIDI
.smi	SMIL Document
.smil	SMIL Document
.snd	Amiga OctaMed music
.snd	AU Audio
.snd	ULAW Audio
.spc	Animated NeoChrome
.spl	FutureSplash Player
.sr	Animated NeoChrome
.ssm	Standard Streaming Metafile
.stm	HTML web page with server side include
.sun	Animated NeoChrome
.sup	Animated NeoChrome
.svx	Amiga OctaMed music
.swf	Shockwave Flash
.tar	Unix Tape Archive
.targa	OS/2 Bitmap/Targa Truevision Image
.taz	Unix Compressed (.z) Files
.tcl	TCL Program
.tex	TeX Document
.texi	GNU TeXinfo Document
.texinfo	GNU TeXinfo Document
.text	Text File
.tga	OS/2 Bitmap/Targa Truevision Image
.tgz	GZIP File
.tif	TIFF Image
.tiff	TIFF Image
.tny	Animated NeoChrome
.ttc	OpenType Font

Table continued on following page

Extension	Description
.ttf	OpenType Font
.txt	Text File
.ul	AU Audio
.ulw	AU Audio
.url	URL Bookmark
.uu	UUEncoded Data
.uue	UUEncoded Data
.vbs	VB Script in text file/MPEG Video
.vcf	VCard
.vew	Lotus Approach Document
.vff	Animated NeoChrome
.vfw	Microsoft Video
.vga	OS/2 Bitmap
.vob	DV Video
.voc	Amiga OctaMed music
.w51	WordPerfect PC 5.1 Doc
.waf	Website Archive
.wav	WAV Audio
.web	Compiled JavaScript and HTML, ready to be served by NES
.wiz	Word Document
.wk1	Lotus Spreadsheet r2.1
.wk3	Lotus 123 Document
.wk4	Lotus 123 Document
.wks	Lotus Spreadsheet r2.1
.wmf	Windows MetaFile image
.wp	WordPerfect PC 5.1 Doc
.wp4	WordPerfect PC 4.2 Doc
.wp5	WordPerfect PC 5.1 Doc
.wp6	WordPerfect PC 4.2 Doc
.wpd	WordPerfect Document
.wpg	Animated NeoChrome
.wpm	WordPerfect PC 4.2 Doc
.wrl	VRML File
.ws	Windows Script File (beta versions of WSH)
.wsf	Windows Script File used by WSH
.wsh	WSH control file
.wve	Amiga OctaMed music
.x10	X-Windows Dump

Table continued on following page

Extension	Description
.x11	X-Windows Dump
.xbm	X-Windows Bitmap
.xlc	Excel Worksheet
.xlm	Excel Worksheet
.xls	Excel Worksheet
.xls	Microsoft Excel Worksheet
.xlt	Excel Worksheet
.xlw	Excel Worksheet
.xm	669 MOD Music
.xml	XML (Extensible Markup Language) Document
.xpm	X-Windows Pixmap
.xsl	XML style sheet Document
.xwd	X-Windows Dump
.z	Unix Compressed Files
.zip	ZIP Archives
.zoo	Zoo Archive

Warnings:

- Beware of the `.pac` file extension. It is a `proxy.pac` auto-config file but it also represents an obscure image format supported by some plugins.

See also:	`.jar`, `.java`, `.js`, `<SCRIPT ARCHIVE="...">`, `<SCRIPT SRC="...">`, `external.AddChannel()`, Host environment, HTML file, Platform, `proxy.pac`

File object (Object/JScript)

A special JScript object representing a file on a locally mounted drive.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`myFile = File`
	IE	*`myFile`* `= new File(`*`aName`*`)`
	IE	*`myFile`* `= `*`myFileSystem`*`.GetFile(`*`aName`*`)`
	IE	*`myFile`* `= `*`myFileSystem`*`.CreateTextFile(`*`aName`*`, `*`aFlag`*`)`
	IE	*`myFile`* `= `*`myFileSystem`*`.OpenTextFile(`*`aName`*`, `*`aMode`*`, `*`aFlag`*`, `*`aFormat`*`)`
Argument list:	*`aName`*	The name of a file
	`aFlag`	A flag indicating whether the file can be overwritten
	`aMode`	An I/O mode for the file
	`aFormat`	A format code indicating ASCII or Unicode content

Object properties:	`Attributes, constructor, dataFld, dataSrc, DateCreated, DateLastAccessed, DateLastModified, defaultValue, Drive, Name, ParentFolder, Path, prototype, recordNumber, ShortName, ShortPath, Size, Type, value`
Object methods:	`blur(), byteToString(), clearError(), click(), close(), Copy(), Delete(), eof(), error(), exists(), flush(), focus(), getLength(), getPosition(), Move(), open(), OpenAsTextStream(), read(), readByte(), readln(), select(), setPosition(), stringToByte(), write(), writeByte(), writeln()`
Event handlers:	`onBlur, onClick, onDblClick, onFocus`

This object is available in many contexts. On the server-side, it allows access to the server file system. In that context it is available as part of the Netscape Enterprise Server product (see the separate `File` object entries for the NES version).

`File` objects may also exist client-side. Similar file system access may be possible there as well, although you should not expect this to work from within your browser. It may be available as part of a desktop scripting environment.

Although they aren't instantiated as `File` objects, it is sometimes convenient to refer to a `File` object when we really mean to refer to an `Input` object whose `type` property is set to the "`FILE`" value.

Warnings:

- Be aware that `File` objects are not standardized and may offer different methods, properties and functions in each context. Furthermore, the same named methods, properties and functions may not yield the same results across all implementations.
- The JScript `File` object and the identically named Netscape Enterprise Server `File` object do not share any properties or methods. They are completely different implementations.
- The JScript `File` object is somewhat odd in that its properties and methods are spelled with a capital letter at the start of their name. This is not typical JavaScript or JScript usage so you should beware of capitalization when scripting `File` objects with JScript.

See also:	`FileSystem` object, `FileSystem.CreateTextFile()`, `FileSystem.GetFile()`,`FileSystem.OpenTextFile()`,`Folder` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`Attributes`	-	3.0 +	-	4.0 +	-	-	-
`dataFld`	-	3.0 +	-	4.0 +	-	-	Warning
`dataSrc`	-	3.0 +	-	4.0 +	-	-	Warning
`DateCreated`	-	3.0 +	-	4.0 +	-	-	-
`DateLastAccessed`	-	3.0 +	-	4.0 +	-	-	-
`DateLastModified`	-	3.0 +	-	4.0 +	-	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
DefaultValue	-	3.0 +	-	4.0 +	-	-	Warning
Drive	-	3.0 +	-	4.0 +	-	-	-
Name	-	3.0 +	-	4.0 +	-	-	-
ParentFolder	-	3.0 +	-	4.0 +	-	-	-
Path	-	3.0 +	-	4.0 +	-	-	-
recordNumber	-	3.0 +	-	4.0 +	-	-	Warning
ShortName	-	3.0 +	-	4.0 +	-	-	-
ShortPath	-	3.0 +	-	4.0 +	-	-	-
Size	-	3.0 +	-	4.0 +	-	-	-
Type	-	3.0 +	-	4.0 +	-	-	ReadOnly
value	-	3.0 +	-	4.0 +	-	-	Warning

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
blur()	-	3.0 +	-	4.0 +	-	-	Warning
click()	-	3.0 +	-	4.0 +	-	-	Warning
Copy()	-	3.0 +	-	4.0 +	-	-	-
Delete()	-	3.0 +	-	4.0 +	-	-	-
focus()	-	3.0 +	-	4.0 +	-	-	Warning
Move()	-	3.0 +	-	4.0 +	-	-	-
OpenAsTextStream()	-	3.0 +	-	4.0 +	-	-	-
select()	-	3.0 +	-	4.0 +	-	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	HTML	Notes
onBlur	-	3.0 +	-	4.0 +	-	-	Warning
onClick	-	3.0 +	-	4.0 +	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	4.0 +	Warning
onFocus	-	3.0 +	-	4.0 +	-	-	Warning

File.Attributes (Property)

The file-system attributes of a file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	Number primitive	
JavaScript syntax:	IE	*myFile*.Attributes

The `File` object is available for client-side use with the MSIE browser. This property contains its file system attributes.

This property manages the attributes as a bit-mask with individual bits controlling different attributes. The bits can be accessed individually using the integer value corresponding to a power of 2. The table lists the integers that represent each different attribute:

Value	Attribute
0	No special attributes –normal file.
1	Read only access.
2	Hidden file.
4	Indicates a system file.
8	Refers to drive volume label and cannot be altered.
16	Refers to a folder and cannot be changed.
32	File has changed and needs to be backed up again.
64	File object represents a shortcut and not a real file.
128	File is compressed.

You can use Bitwise OR expressions to merge them or accomplish the same with integer additions.

You cannot alter the settings of bits 8, 16, 64 and 128 as these affect the structure of a file. That is to say, you cannot change a file into a folder or disk volume.

You should read the current attributes setting and then modify it to write it back. The example illustrates some simple functions that encapsulate this conveniently.

Where the bit needs to be set, a simple bitwise OR with a single bit value is accomplished in a single line. To clear a bit, we could use a bitwise AND having the corresponding bit clear. In these examples a different technique is used for illustration where the bit is set regardless of its previous state and is then cleared using a subtraction. That saves the computation of a complex bit-mask. An intermediate temporary variable is used to avoid signalling the operating system with unnecessarily modification requests.

There are other alternative ways to accomplish this and you could write some generic functions to examine, set or clear a bit in a bit-mask and then call them from each of these wrappers indicating the bit you want to operate on.

Example code:

```
// Examine the read/write flag
function isReadOnly(aFile)
{
   return Boolean(aFile.Attributes & 1);
}
// Set the file read only
function setReadOnly(aFile)
{
   aFile.Attributes |= 1;
}
```

```
// Set the file read/write
function setWriteOnly(aFile)
{
   var myAttributes = aFile.Attributes |= 1;
   aFile.Attributes = myAttributes - 1;
}
// ----------------------------------------------------
// Examine the hidden flag
function isHidden(aFile)
{
   return Boolean(aFile.Attributes & 2);
}
// Hide the file
function setHidden(aFile)
{
   aFile.Attributes |= 2;
}
// Reveal the file
function setVisible(aFile)
{
   var myAttributes = aFile.Attributes |= 2;
   aFile.Attributes = myAttributes - 2;
}
// ----------------------------------------------------
// Examine the system flag
function isSystemFile(aFile)
{
   return Boolean(aFile.Attributes & 4);
}
// Set file to be a system file
function setSystem(aFile)
{
   aFile.Attributes |= 4;
}
// Set file to be a non system file
function setPublic(aFile)
{
   var myAttributes = aFile.Attributes |= 4;
   aFile.Attributes = myAttributes - 4;
}
// ----------------------------------------------------
// Examine the drive volume flag
function isDriveVolume(aFile)
{
   return Boolean(aFile.Attributes & 8);
}
// ----------------------------------------------------
// Examine the folder flag
function isFolder(aFile)
{
   return Boolean(aFile.Attributes & 16);
}
// ----------------------------------------------------
// Examine the backup flag
function needsBackup(aFile)
{
```

```
   return Boolean(aFile.Attributes & 32);
}
// Set backup required
function setBackup(aFile)
{
   aFile.Attributes |= 32;
}
// Clear backup
function clearBackup(aFile)
{
   var myAttributes = aFile.Attributes |= 32;
   aFile.Attributes = myAttributes - 32;
}
// ------------------------------------------------
// Examine the shortcut flag
function isShortCut(aFile)
{
   return Boolean(aFile.Attributes & 64);
}
// ------------------------------------------------
// Examine the compressed flag
function isCompressed(aFile)
{
   return Boolean(aFile.Attributes & 128);
}
```

See also:	`Folder.Attributes`

File.Copy() (Method)

A method that copies files.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFile*.Copy(*newFileName*, *aFlag*)
Argument list:	*newFileName*	The new file name to copy the file to
	aFlag	An indication of whether to overwrite or not

When operating on files on the client platform with MSIE, this method provides a means of copying files within the file system.

The destination argument needs to be a valid location within the file system the `File` object belongs to. Given that computers allow for the mounting of foreign file systems through a variety of different techniques, the target file system path may be constructed quite differently to that of the source path.

See also:	`FileSystem.CopyFile()`,`Folder.Copy()`

File.DateCreated (Property)

The creation date of the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFile`*`.DateCreated`

The date the file was first created. This is obtained by inspecting the file system and directory information for the file.

See also:	`Folder.DateCreated`

File.DateLastAccessed (Property)

The date that a file was last accessed.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFile`*`.DateLastAccessed`

The date that the file was last accessed by an application. This is obtained by inspecting the file system and directory information for the file.

See also:	`Folder.DateLastAccessed`

File.DateLastModified (Property)

The last modified data for a file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFile`*`.DateLastModified`

The date that the file was last modified by being written to. This is obtained by inspecting the file system and directory information for the file.

See also:	`Folder.DateLastModified`

File.Delete() (Method)

A method that deletes files.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFile*.Delete()

If need be, you can delete a file from the file system in the client environment. This may not be possible from a web browser due to the security protections surrounding the browser.

However, JScript is available as a general purpose scripting tool as part of the WSH facility on Windows. In that context it may well need to be able to delete a file.

File.Drive (Property)

The drive name that the file is stored on.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myFile*.Drive

This returns the drive name describing the drive that the file is currently stored on.

See also:	Folder.Drive

File.Move() (Method)

A file move method.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFile*.Move(*aPath*)
Argument list:	*aPath*	The new target part for the file

You may use this method in the Windows environment to move a file to a different folder within the file system. This follows the Unix tradition of calling the Rename command a Move command.

The destination path should be valid for the file system which the file is being moved. Given that computers allow for the mounting of foreign file systems through a variety of different techniques, the target file system path may be constructed quite differently to that of the source path.

See also:	FileSystem.MoveFile(),Folder.Move()

File.Name (Property)

The name of the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFile`*`.Name`

The name portion of the file's full path and name is returned by this property.

You can also change the contents of this property to rename the file without moving it from the folder it is currently kept in.

See also:	`Folder.Name`

File.OpenAsTextStream() (Method)

A means of opening the file as if it were a text stream so JScript can do I/O on it.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`TextStream object`	
JavaScript syntax:	IE	`myFile.OpenAsTextStream(aMode, aFormat)`
Argument list:	*`aMode`*	An access mode for the file
	`aFormat`	A format control for the file

This method opens the file for JScript so that it can access or modify its contents when the JScript interpreter is in the Windows environment.

The mode parameter should be one of:

- `ForReading`
- `ForWriting`
- `ForAppending`

The format parameter can be set to open the file with a Unicode or ASCII character set. The default is ASCII if you request the system default value. These values are indicated with a numeric value as follows:

Value	Format
`-2`	Use system default
`-1`	Unicode text format
`0`	ASCII text format

See also:	`FileSystem.OpenTextFile()`, `TextStream` object

File.ParentFolder (Property)

The parent folder that contains the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Folder object`	
JavaScript syntax:	IE	*`myFile`*`.ParentFolder`

The `File` object represents a file living within a folder in the file system. This property returns an object that represents the folder containing the file.

See also:	`Folder` object, `Folder.ParentFolder`, `Folder.SubFolders[]`

File.Path (Property)

The path within the file system that locates the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFile`*`.Path`

The full path to reach the file within the file system is yielded by this property.

See also:	`Folder.Path`

File.ShortName (Property)

The DOS 8.3 filename for the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myFile.ShortName`

In the Windows environment, even though long file names are used from the user's point of view, at the lowest level within the file system some files will still be stored under their old style DOS 8.3 format file names. This property yields the DOS equivalent short file name for a `File` object.

See also:	`Folder.ShortName`

File.ShortPath (Property)

The DOS 8.3 compatible path to the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFile`*`.ShortPath`

In the Windows environment, even though long file and folder names are used from the user's point of view, at the lowest level within the file system some folders and files will still be stored under their old style DOS 8.3 format file names. This property yields the DOS equivalent path name for a `File` object.

See also:	`Folder.ShortPath`

File.Size (Property)

The size of the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myFile`*`.Size`

The size of the file measured in bytes is yielded by this property. This is effectively the same as the value returned by the `getLength()` method in the NES server environment.

See also:	`Folder.Size`

File.Type (Property)

The type of the file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFile`*`.type`

The file type extension of the file is yielded by this property. In the past, extensions conformed to a three character standard. With the cross mounting of foreign file systems which did not conform to the DOS 8.3 standard, file extensions can be of any length although they are usually less than five characters long.

It is normal to hide the file extensions from the users of desktop systems. In that case you may want to hold the filename and extension separately, so that you can hide the file extension to be consistent with what your users see on the desktop.

See also:	`Folder.Type`, `Input.type`

Property attributes:

`ReadOnly`.

File object (Object/NES)

An object that encapsulates a file on the local file system within the server.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
JavaScript syntax:	NES	*myFile* = File
	NES	*myFile* = new File(*aName*)
Argument list:	*aName*	A valid file name
Object methods:	byteToString(), clearError(), close(), eof(), error(), exists(), flush(), getLength(), getPosition(), open(), read(), readByte(), readln(), setPosition(), stringToByte(), write(), writeByte(), writeln()	

You can create a new instance of this class with the `File()` constructor.

Files tend to behave in a somewhat similar way in all languages and implementations. This generally means that they will read and write in 8 bit character sized elements. Beware of this as JavaScript is Unicode based so file contents may not exactly mirror the in-memory representation of a string.

Warnings:

- Be careful not to confuse this with the JScript `File` object. The JScript object may be supported in a Microsoft server but the Netscape Enterprise Server supports an object with a mutually exclusive set of properties and methods and neither kind of `File` object has any chance of being portable between a Netscape Enterprise Server and a Microsoft server. There are so many other differences between their object models that it is a waste of time trying to write a portable server-side script.

See also:	Netscape Enterprise Server, `unwatch()`, `watch()`

Method	JavaScript	JScript	NES	Notes
byteToString()	1.1 +	-	2.0 +	-
clearError()	1.1 +	-	2.0 +	-
close()	1.1 +	-	2.0 +	-
eof()	1.1 +	-	2.0 +	-

Method	JavaScript	JScript	NES	Notes
error()	1.1 +	-	2.0 +	-
exists()	1.1 +	-	2.0 +	-
flush()	1.1 +	-	2.0 +	-
getLength()	1.1 +	-	2.0 +	-
getPosition()	1.1 +	-	2.0 +	-
open()	1.1 +	-	2.0 +	-
read()	1.1 +	-	2.0 +	-
readByte()	1.1 +	-	2.0 +	-
readln()	1.1 +	-	2.0 +	-
setPosition()	1.1 +	-	2.0 +	-
stringToByte()	1.1 +	-	2.0 +	-
write()	1.1 +	-	2.0 +	-
writeByte()	1.1 +	-	2.0 +	-
writeln()	1.1 +	-	2.0 +	-

File() (Constructor)

A constructor for creating new instances of the `file` object in an NES server.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`File object`	
JavaScript syntax:	NES	`new File(`*`aFileName`*`);`
Argument list:	*`aFileName`*	A file path and name that is valid within the local file system

This constructor is used for creating new `File` objects, like this:

```
myFileObject = new File("/some/path/to/a/folder/filename.txt");
```

File.byteToString() (Method)

Convert a byte value into a string.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`String object`	
JavaScript syntax:	NES	*`myFile`*`.byteToString(`*`aNumber`*`)`
Argument list:	*`aNumber`*	A byte value to be converted to a string

This method provides a way of converting a single numeric value that represents a code point in the native character set to a character than can be written to a file from a string variable.

There is other similar functionality supported as a static method of the `String` object.

See also:	`File.stringToByte()`,`String.fromCharCode()`

File.clearError() (Method)

Reset the error property value.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
JavaScript syntax:	NES	*myFile*`.clearError()`

If the file access caused an error, you can clear that error status value by calling this method.

See also:	`File.error()`

File.close() (Method)

Close the file that was opened with the `File.open()` method.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	*myFile*`.close()`

This method closes a file that was opened for reading, writing or appending. It returns a Boolean value to indicate the success or failure of the closure.

See also:	`File.open()`

File.constructor (Property)

A reference to the constructor function for this object.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`Function object`	
JavaScript syntax:	NES	*myFile*`.constructor`

The constructor is that of the built-in `File` prototype object.

You can use this as one way of creating file objects although it is more popular to use the new `File()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

File.eof() (Method)

This method returns a flag indicating whether we are at the end of the file or not.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	*`myFile`*`.eof()`

If we are currently positioned at the end of the file (the `getLength()` and `getPosition()` method calls return an equivalent value), then this method returns the Boolean `true` value. Any other position within the file will return `false`.

A zero length file can only ever have a file pointer positioned at its end of file so it will always return `true` if the `getLength`() method returns zero.

File.error() (Method)

An accessor function for the error property.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myFile`*`.error()`

If an error occurs during an operation on a `File` object, this property will yield the value of that error. You will need some additional source of reference to make sense of the possible error codes that might be presented by this method, as they will be operating system specific.

See also:	`File.clearError()`

File.exists() (Method)

A method that returns a flag to indicate whether the file exists or not.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	`myFile.exists()`

This method returns the Boolean `true` value if the file already exists within the file system and a Boolean `false` value if it does not.

File.flush() (Method)

Purges the output buffer to the file after some writing.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
JavaScript syntax:	NES	*`myFile`*`.flush()`

File writing tends to involve a certain amount of buffering by the file manager in the environment in which you are accessing the files. This is necessary to improve performance and throughput. The data is only physically transferred to the file when a buffer full of data is ready or when the file buffer is closed. This can mean that a JavaScript error can leave the file incomplete if it fails in a way that prevents the file from being closed properly. A run-time error would normally not write any pending data out to a file.

The `flush()` method allows you to force the file contents to be updated so that the file is complete and there are no pending contents yet to be written. You might force a `flush()` at the end of a record for example.

A `flush()` method may be called for much more frequently if you are using fixed length records and manually maintaining an index structure at the front of the file.

See also:	`File.write(),File.writeByte(),File.writeln(), response.flush()`

File.getLength() (Method)

Returns the length of the file measured in 8 bit characters.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*`myFile`*`.getLength()`

The length of the file is returned. You may want to compute a position within the file based on this value and use it with the `setPosition()` method later on.

See also:	`File.setPosition()`

File.getPosition() (Method)

Returns the current position in the file where the next read or write is to take place.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*myFile*`.getPosition()`

The position of the file pointer within the file is returned relative to the beginning of the file. You can then use this value with the `setPosition()` method.

See also:	`File.setPosition()`

File.open() (Method)

Opens a file for reading, writing or appending.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
JavaScript syntax:	NES	*myFile*`.open(`*aMode*`);`
Argument list:	*aMode*	A mode of opening the file (read, write, append)

This method opens the file that is encapsulated by the `File` object.

C language programmers will be immediately familiar with this method. It behaves very similarly to the `fopen()` call in the ANSI C language libraries. The difference is that this method lacks a filename argument which is moved to the constructor function which is called separately.

The following flag values can be used with this method:

Flag	Description
r	Opens the file for reading
r+	Opens the file for reading and writing
w	Opens the file for writing
w+	Opens the file for writing and reading

Table continued on following page

Flag	Description
A	Opens the file for appending
a+	Opens the file for reading and appending
br	Opens a Windows binary file for reading
br+	Opens a Windows binary file for reading and writing
bw	Opens a Windows binary file for writing
bw+	Opens a Windows binary file for writing and reading
ba	Opens a Windows binary file for appending
ba+	Opens a Windows binary file for reading and appending

If you are using this in a server-side application (within NES), you should make sure the project locking is activated to avoid file corruption happening if there are multiple simultaneous accesses to the file.

Example code:

```
<SERVER>
// Example file create, open and write taken from
// Wrox Professional JavaScript
myFile = new File("file.txt");
myFile.open("a");
myFile.writeln("Append this line to the file.");
myFile.close();
</SERVER>
```

See also:	`File.close()`,`FileSystem.CreateTextFile()`, `FileSystem.OpenTextFile()`,`project.lock()`

File.prototype (Property)

The prototype for the `File` object which can be used to extend the interface for all `File` objects.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`File object`	
JavaScript syntax:	NES	`File.prototype`
	NES	*`myFile`*`.constructor.prototype`

Refer to:

`prototype` property

File.read() (Method)

Reads a string of characters from the file.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*myFile*`.read(`*aByteCount*`)`
Argument list:	*aByteCount*	Read this many bytes from the file

When you call this method, you need to indicate the length of the string you want to retrieve.

You might use this in a record structured file where the records are all of a fixed length.

See also:	`File.readByte(),File.readln(),File.write()`

File.readByte() (Method)

Reads a single byte or character from the file.

Availability:	JavaScript –1.1 Netscape Enterprise Server version –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*myFile*`.readByte()`

When this reads a byte, the position register for the `File` object is indexed onwards by one item.

This would be used to read a file as a stream of characters where the individual records will need to be determined by the script as it reads and buffers the file.

See also:	`File.read(),File.readln(),File.writeByte()`

File.readln() (Method)

Reads from the current position up to the next newline character in the file.

Availability:	JavaScript –1.1 Netscape Enterprise Server version –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*myFile*`.readln()`

After this method returns the string read from the file, the position pointer is then located at the beginning of the next line.

You would use this to read records that are variable length and terminated by a newline character.

See also:	`File.read()`,`File.readByte()`

File.setPosition() (Method)

Sometimes you may need to reposition the file pointer. This method allows you to position it at exactly the right place.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	*`myFile`*`.setPosition(`*`aPosition`*`)`
	NES	*`myFile`*`.setPosition(`*`aPosition`*`, `*`aControl`*`)`
Argument list:	*`aControl`*	A means of indicating what kind of offset to use
	`aPosition`	A valid position within the file

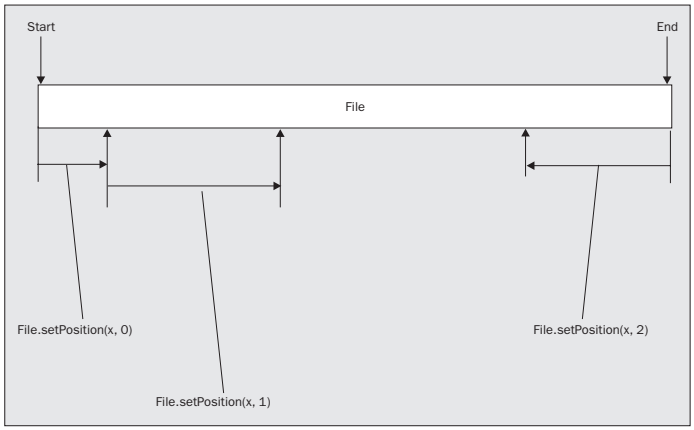
File positioning mechanisms are a fundamental part of building indexed record oriented files and managing fixed length records and fields in a sequential and variable length file structure.

The position control takes an integer value that selects a reference technique for setting the new position value.

A position control value of `0` (zero) indicates the new position should be measured from the start of the file.

A position control value of `1` adds the offset to the current position. This should allow forwards or backwards movement relative to the current position.

A position control value of `2` is measured relative to the end of the file.



See also:	`File.getLength()`,`File.getPosition()`

File.stringToByte() (Method)

Converts the first character of the string argument into a byte value.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*myFile*`.stringToByte(`*aString*`)`
Argument list:	*aString*	A string containing at least one character

This converts a single character to a numeric value. It is equivalent to the character to number conversions supported by the `String` object.

See also:	`File.byteToString()`,`String.charAt()`, `String.charCodeAt()`

File.write() (Method)

Writes a string of data out to the file.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	*myFile*`.write(`*aString*`)`
Argument list:	*aString*	Some text to be written to the file

As the string is written to the file, the method will return a Boolean value that indicates success or failure of the write access.

The length of the content is the controlling factor here. This might be used to write individual fields of a variable length record structure or, if you have loaded the string value you intend to write with a fixed length value, you can use this to write fixed length data to a file.

See also:	`File.flush()`,`File.read()`,`File.writeByte()`, `File.writeln()`

File.writeByte() (Method)

Writes a single byte to the file.

Availability:	JavaScript –1.1 Netscape Enterprise Server–2.0	
JavaScript syntax:	NES	*myFile*.writeByte(*aNumber*)
Argument list:	*aNumber*	The value of a byte to be written to the file

This is the complement of the `readByte()` method which is used to write a continuous stream of character bytes to a file. This imposes no structure on the file other than that which you contrive with your JavaScript code.

See also:	`File.flush()`,`File.readByte()`,`File.write()`, `File.writeln()`

File.writeln() (Method)

Writes a string and automatically place a newline character after it in the file.

Availability:	JavaScript –1.1 Netscape Enterprise Server version –2.0	
JavaScript syntax:	NES	*myFile*.writeln(*aString*)
Argument list:	*aString*	Some text to be written to the file

This method is functionally very similar to the `write()` method. In this case however, an additional trailing newline character is added automatically by the method. You could append the newlines yourself and use the `write()` method instead.

Use this technique to write variable length records to a file.

See also:	`File.flush()`,`File.write()`,`File.writeByte()`

file: URL (Request method)

Loads a local file into a window.

This request method is useful for retrieving files from the local file systems or can be used to display a window containing the HTML version of a directory in the client machine.

The file path values are platform dependent although web browsers tend to understand Unix path rules and convert them to the local platform specific conventions, although MSIE doesn't do this quite as well when navigating local file systems.

On MSIE for Windows, you must at least specify a disk drive letter because the Unix root directory is not mapped. This presents network mapped drives and removable media all at the same peer level. So start with this:

```
file:c:\
```

MSIE is so well integrated into the Windows environment that it immediately recognizes that you are browsing the file system and goes into desktop explorer mode. This adds a folder icon to the toolbar and you can then navigate exactly as you would from the desktop.

The web browsers attempt to take this functionality across to other platforms and so on the Macintosh, with MSIE, try this:

```
file:/
```

MSIE version 5.0 does a much better job and is aware of local network zones and can see other machines connected via the AppleTalk protocol. Given that you can satisfy the security requirements, you can browse a network of machines and keep shared documents on a workgroup server. On the Macintosh MSIE does a better job if the Finder has already mounted a shared volume on the desktop.

Using this technique you can also run applications provided the URL resolves to the name of an executable file. On Windows 98, this URL fires up the desktop calculator:

```
file:C:\Windows\Calc.exe
```

This means you can build a link on your page that activates local applications really easily. This works across platforms too. On the Macintosh, a functionally equivalent URL would be:

```
file://G3/APPS/Desk%20tools/Calculator%207.5
```

You may need to explore your hard disk to work out the paths to your installed applications. Don't forget that you need to use escaped URL values.

The same automated activation works for documents too. For example, locate an Excel spreadsheet and link to it from a document. When you click on the link, this will open Excel and load the document into it.

Older versions of Netscape Navigator are less capable of navigating the whole file system and initially start at the folder the browser application lives in. Netscape 6.0 exhibits a bug and this whole area of the browser needs more work before it is useful.

Accessing these capabilities from script is a little more tricky. `window.location.href` assignment effectively does the same thing on a Macintosh. On Windows, there are already capabilities for activating applications via the ActiveXObject and you need to use that technique because `window.location.href` does not like having applications loaded into browser frames.

Warnings:

- Be very careful what you browse on your system and how. You may corrupt your system although read-only access is unlikely to cause any harm.

See also:	javascript: URL, URL, view-source: URL

Files object (Object/JScript)

A collection of files belonging together in a folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFiles* = *myFolder*.Files
Object properties:	Count	
Object methods:	Item()	

This object is obtained by inspecting the `Files` property of a `Folder` object. The `Files` object is a collection containing objects that encapsulate the files in that folder. Each file is represented by a separate `File` object. You can then examine each of those objects to operate on them as you need to.

The `Files` object provides a method for searching and extracting a named `File` object within the collection and a property that yields the number of `File` objects in the collection.

You should be able to traverse the `Files` object using array indexing techniques or by means of an `Enumerator` object.

See also:	`Enumerator` object

Property	JavaScript	JScript	N	IE	Opera	Notes
Count	-	3.0 +	-	4.0 +	-	ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	Notes
Item()	-	3.0 +	-	4.0 +	-	-

Files.Count (Property)

A count of the number of file items in the `Files` collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	Number primitive	
JavaScript syntax:	IE	*myFiles*.Count

The number of `File` objects belonging to the `Files` collection is yielded by this property. You can build enumeration loops if you know the extent of a collection. You cannot change this value.

Property attributes:

`ReadOnly`.

Files.Item() (Method)

An accessor for retrieving individual `File` objects by name from a `Files` collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`File object`	
JavaScript syntax:	IE	*`myFiles`*`.Item(`*`aName`*`)`
Argument list:	*`aName`*	The name of a file within its folder

With this method, you can locate a named `File` object within the `Files` collection if it is present. The file name is used as a key to locate the required object.

FileSystem object (Object/JScript)

An object that represents an entire file system for a specific disk drive.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*`myFileSystem`* `= ` *`myDrive`*`.FileSystem`
	IE	*`myFileSystem`* `= new ActiveXObject ("Scripting.FileSystemObject")`
Object methods:	`BuildPath(), CopyFile(), CopyFolder(), CreateFolder(), CreateTextFile(), DeleteFile(), DeleteFolder(), DriveExists(), FileExists(), FolderExists(), GetAbsolutePathName(), GetBaseName(), GetDrive(), GetDriveName(), GetExtensionName(), GetFile(), GetFileName(), GetFolder(), GetParentFolderName(), GetSpecialFolder(), GetTempName(), MoveFile(), MoveFolder(), OpenTextFile()`	
Collections:	`Drives[]`	

When building scripts to run in the Windows environment, possibly for use with WSH, you may need to operate on files and folders within the file system hierarchy of the computer. This object encapsulates the file system so that you can operate on its methods and properties.

File systems may have been shared across from other computers but the network manager should make them appear identical to local drives. Foreign file systems should also be mapped to local drives and appear to have a structure that is familiar. However, there may be file name differences. Certain characters may be valid on one operating system and invalid on another or there may be different limitations on file name length and capitalization rules.

See also:	`Drive.FileSystem`, `File` object

Method	JavaScript	JScript	N	IE	Opera	Notes
`BuildPath()`	-	3.0 +	-	4.0 +	-	-
`CopyFile()`	-	3.0 +	-	4.0 +	-	-
`CopyFolder()`	-	3.0 +	-	4.0 +	-	-
`CreateFolder()`	-	3.0 +	-	4.0 +	-	-
`CreateTextFile()`	-	3.0 +	-	4.0 +	-	-
`DeleteFile()`	-	3.0 +	-	4.0 +	-	-
`DeleteFolder()`	-	3.0 +	-	4.0 +	-	-
`DriveExists()`	-	3.0 +	-	4.0 +	-	-
`FileExists()`	-	3.0 +	-	4.0 +	-	-
`FolderExists()`	-	3.0 +	-	4.0 +	-	-
`GetAbsolutePath Name()`	-	3.0 +	-	4.0 +	-	-
`GetBaseName()`	-	3.0 +	-	4.0 +	-	-
`GetDrive()`	-	3.0 +	-	4.0 +	-	-
`GetDriveName()`	-	3.0 +	-	4.0 +	-	-
`GetExtensionName()`	-	3.0 +	-	4.0 +	-	-
`GetFile()`	-	3.0 +	-	4.0 +	-	-
`GetFileName()`	-	3.0 +	-	4.0 +	-	-
`GetFolder()`	-	3.0 +	-	4.0 +	-	-
`GetParentFolderName()`	-	3.0 +	-	4.0 +	-	-
`GetSpecialFolder()`	-	3.0 +	-	4.0 +	-	-
`GetTempName()`	-	3.0 +	-	4.0 +	-	-
`MoveFile()`	-	3.0 +	-	4.0 +	-	-
`MoveFolder()`	-	3.0 +	-	4.0 +	-	-
`OpenTextFile()`	-	3.0 +	-	4.0 +	-	-

FileSystem.BuildPath() (Method)

A method for manufacturing paths within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myFileSystem*`.BuildPath(`*aPath*`, `*aName*`)`
Argument list:	*aPath*	A previously existing path
	aName	A name to be added to extend the path

This yields a path value for use within the file system.

FileSystem.CopyFile() (Method)

A method for copying files within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFileSystem*`.CopyFile(`*aSource*`, `*aTarget*`, `*aFlag*`)`
Argument list:	*aSource*	The file to copy from
	aTarget	The file to copy to
	aFlag	A flag to indicate whether to overwrite

Although you can copy files by using the `Copy()` method that belongs to a `File` object, the `FileSystem` object also supports file copying from a point of view that is external to a file.

See also:	`File.Copy()`

FileSystem.CopyFolder() (Method)

A method for copying folders within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFileSystem*`.CopyFolder(`*aSource*`, `*aTarget*`, `*aFlag*`)`
Argument list:	*aSource*	The folder to copy from
	aTarget	The folder to copy to
	aFlag	A flag to indicate whether to overwrite

From the `FileSystem`, you can copy a whole folder hierarchy in a single operation. As is the case with the `File` object, the `Folder` object also provides a `Copy()` method too.

See also:	`Folder.Copy()`

FileSystem.CreateFolder() (Method)

A method for creating new folders within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Folder object`	
JavaScript syntax:	IE	*myFileSystem*`.CreateFolder(`*aName*`)`
Argument list:	*aName*	The name of a folder to be created

You may need to create a new folder as a destination to copy files to or to use as a location to create them in.

See also:	`Folder` object

FileSystem.CreateTextFile() (Method)

A method for creating new empty text files within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`File object`	
JavaScript syntax:	IE	*myFileSystem*`.CreateTextFile(`*aName*`,` *aFlag*`)`
Argument list:	*aName*	The name of a text file to be created
	aFlag	A flag to indicate whether to overwrite

This is a means of creating a new text file. As is often the case with the Windows environment, it has become bloated by having several alternative methods for achieving the same result. If you are not careful, this can lead to sloppy coding and some very difficult to maintain systems.

It is highly recommended that you choose only one of the several available techniques where there are alternatives and apply that wherever you can. It is likely you'll find one of the other ways of creating new files more appropriate and you should be able to ignore this method most of the time.

See also:	`File` object, `File.open()`

FileSystem.DeleteFile() (Method)

A method for deleting files from the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFileSystem*.DeleteFile(*aName*, *aFlag*)
Argument list:	*aName*	The name of a file to be deleted
	aFlag	A flag that indicates whether to force the deletion or not

Files can be deleted by using the `File` object's own `Delete()` method or by deleting them at the file system level.

See also:	`File.Delete()`

FileSystem.DeleteFolder() (Method)

A method for deleting folders from the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFileSystem*.DeleteFolder(*aName*, *aFlag*)
Argument list:	*aName*	The name of a folder to be deleted
	aFlag	A flag that indicates whether to force the deletion or not

As is the case with files, there is an alternative way to delete folders too. You can either use this method or the `Delete()` method that belongs to the `Folder` object itself.

See also:	`Folder.Delete()`

FileSystem.DriveExists() (Method)

A method for testing the existence of a drive.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myFileSystem*.DriveExists(*aName*)
Argument list:	*aName*	The name of a disk drive to be tested

You can test for the existence of a drive within the file system using this method.

FileSystem.Drives[] (Collection)

A list of all `Drive` objects within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Drives object`	
JavaScript syntax:	IE	*`myFileSystem`*`.Drives`

This property yields a collection of `Drive` objects, each one representing a different drive unit.

See also:	`Drives` object

FileSystem.FileExists() (Method)

A means of testing for the existence of a file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.FileExists(`*`aName`*`)`
Argument list:	*`aName`*	The name of a file to be tested for existence

You can test for the existence of a file in the file system with this method.

FileSystem.FolderExists() (Method)

A means of testing for the existence of a folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.FolderExists(`*`aName`*`)`
Argument list:	*`aName`*	The name of a folder to be tested for existence

You can test for the existence of a folder within the file system with this method.

FileSystem.GetAbsolutePathName() (Method)

A method that returns the complete path name of a file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetAbsolutePathName(`*`aPath`*`)`
Argument list:	*`aPath`*	A relative or absolute path to be examined

Given that you have a partial reference to a file by name, this method will return a fully qualified path that describes that file's unique location within the file system.

FileSystem.GetBaseName() (Method)

A method that returns the base name of a file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetBaseName(`*`aPath`*`)`
Argument list:	*`aPath`*	A path to be examined

This method yields the base portion of a path.

FileSystem.GetDrive() (Method)

A method that returns a drive object from the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Drive object`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetDrive(`*`aName`*`)`
Argument list:	*`aName`*	The name of a drive in the file system

Given a path within the file system, this method returns an object representing the drive containing that path.

See also:	`Drive` object

FileSystem.GetDriveName() (Method)

A method to obtain the drive name of a drive within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetDriveName(`*`aLetter`*`)`
Argument list:	*`aLetter`*	The letter of a drive to examine for its name

Given a file path, the drive name within it is extracted and returned.

FileSystem.GetExtensionName() (Method)

A method to yield the file extension from the path name.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetExtensionName(`*`aPath`*`)`
Argument list:	*`aPath`*	A full path to the file to be examined

Given a path name, the file extension portion of it is extracted and returned.

FileSystem.GetFile() (Method)

A method to obtain a `File` object from a File system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`File object`	
JavaScript syntax:	IE	`myFileSystem.GetFile(aName)`
Argument list:	*`aName`*	The name of a file to construct a `File` object for

Given a file path within the file system, this method returns an object that represents the file.

See also:	`File` object

FileSystem.GetFileName() (Method)

A method to extract a file name from a path.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetFileName(`*`aPath`*`)`
Argument list:	*`aPath`*	A full or relative path to examine

Given a file path, this method extracts and returns the file name portion of it.

FileSystem.GetFolder() (Method)

A method to extract the folder from a path.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Folder object`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetFolder(`*`aPath`*`)`
Argument list:	*`aPath`*	A path to examine for a deepest folder object

Given a path name, this method extracts and returns an object representing the innermost folder within it.

See also:	`Folder` object

FileSystem.GetParentFolderName() (Method)

A method to extract the parent folder from a path.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFileSystem`*`.GetParentFolderName(`*`aPath`*`)`
Argument list:	*`aPath`*	The path to a file or folder whose parent is required

Given a path, this method returns the name of a parent folder within that path.

FileSystem.GetSpecialFolder() (Method)

A method to yield a special folder within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Folder object`	
JavaScript syntax:	IE	*myFileSystem*`.GetSpecialFolder(`*aNumber*`)`
Argument list:	*aNumber*	An identifier for the special folder required

There are many special locations within the file system where objects are stored. Given a keyword, this method maps the keywords to physical locations within the file system.

The following special folders are supported in the Windows environment:

Number	Description
0	Windows files folder
1	System files folder
2	Temporary items folder

See also:	`Folder` object

FileSystem.GetTempName() (Method)

A method to generate a unique temporary file name.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myFileSystem*`.GetTempName()`

There are often times when you need a temporary file name. If you want to generate a unique name yourself you can. It is generally better to let the operating system do this because it can generate and test for uniqueness far more reliably than you can.

FileSystem.MoveFile() (Method)

A method to move a file within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFileSystem*.MoveFile(*aSource*, *aTarget*)
Argument list:	*aSource*	The file to move from
	aTarget	The file to move to

As well as being able to move files with their own `Move()` method, you can move them from within the file system as well.

See also:	`File.Move()`

FileSystem.MoveFolder() (Method)

A method to move a folder within the file system.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFileSystem*.MoveFolder(*aSource*, *aTarget*)
Argument list:	*aSource*	The folder to move from
	aTarget	The folder to move to

Folder objects can be moved from within the file system or by invoking the `Move()` method that belongs to the `Folder` object itself.

See also:	`Folder.Move()`

FileSystem.OpenTextFile() (Method)

A method for opening text files for I/O.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`File object`	
JavaScript syntax:	IE	*myFileSystem*.OpenTextFile(*aName*, *aMode*, *aFlag*, *aFormat*)

Argument list:	*aName*	The name of the file to be created
	aFlag	A flag indicating whether the file can be created if necessary
	aMode	An access mode for the file
	aFormat	A format control for the file

Yet another way to open files that is named inconsistently with every other way that files can be operated on. In this case, we are opening files from the file system's point of view.

It is recommended that you select only one preferred technique for opening text files for I/O and sticking to it. If you mix and match and use several alternatives, you will become confused and so will everyone else who has to maintain your scripts in the future.

This is the equivalent of a `File.Open()` when the JScript interpreter in the Windows environment needs to read the contents of the text in the file.

The mode parameter should be one of:

- `ForReading`
- `ForWriting`
- `ForAppending`

The format parameter can be set to open the file with a Unicode or ASCII character set. The default is ASCII if you request the system default value. This parameter is indicated with a numeric value as follows:

Value	Format
`-2`	Use system default
`-1`	Unicode text format
`0`	ASCII text format

See also:	`File` object, `File.open()`, `File.OpenAsTextStream()`, `TextStream` object

FileUpload object (Object/DOM)

A text field in a form for entering the name of a file to be uploaded to the server.

Availability:	DOM level –1 JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0 Opera –3.0
Inherits from:	`Input object`

JavaScript syntax:	-	*myFileUpload* = *myDocument*.aFormName.anElement Name
	-	*myFileUpload* = *myDocument.aFormName*.elements[*anItemIndex*]
	IE	*myFileUpload* = *myDocument*.all.*anElementID*
	IE	*myFileUpload* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*myFileUpload* = *myDocument*.all[*aName*]
	-	*myFileUpload* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*myFileUpload* = *myDocument*.forms[*aFormIndex*] .elements [*anItemIndex*]
	-	*myFileUpload* = *myDocument*.getElementById(*anElementID*)
	-	*myFileUpload* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myFileUpload* = *myDocument*.getElementsByTagName ("INPUT") [*anIndex*]
HTML syntax:	<INPUT TYPE="file">	
Argument list:	*anItemIndex*	A reference to an element in a collection
	anIndex	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
	aFormIndex	A reference to one specific form in the collection
Object properties:	accept, size, type, value	
Object methods:	handleEvent(), select()	
Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onChange, onDragStart, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onSelect, onSelectStart	

Many properties, methods and event handlers are inherited from the Input object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all subclasses of the Input object superclass.

There isn't really a FileUpload object class, but it is helpful when trying to understand the wide variety of input element types, if we can reduce the complexity by discussing only the properties and methods of a file upload. In actual fact, the object is represented as an item of the Input object class.

Unlike MSIE, Netscape Navigator does not support the defaultValue property or the select() method for this subclass of the Input object.

Warnings:

- To be able to upload a file under script control, you must have the UniversalFileRead privilege granted to the script.

See also:	`Element` object, `FileUpload.handleEvent()`, `Form.elements[]`, `Input` object, `Input.accessKey`, `onChange`, `UniversalFileRead`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`accept`	-	5.0 +	-	5.0 +	-	-	-	-
`size`	1.0 +	3.0 +	2.0 +	4.0 +	-	-	-	-
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-
`select()`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onChange`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onResize`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
`onRowEnter`	-	3.0 +	-	4.0 +	-	-	-	-
`onRowExit`	-	3.0 +	-	4.0 +	-	-	-	-
`onSelect`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

FileUpload.accept (Property)

A list of acceptable MIME types for a file upload in a form.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFileUpload`*`.accept`

Refer to:

MIME types

FileUpload.handleEvent() (Method)

Passes an event to the appropriate handler for this object.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myFileUpload`*`.handleEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An event to be handled by this object

This applies to Netscape Navigator prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`FileUpload` object, `handleEvent()`

FileUpload.select() (Method)

Triggers a `Select` event on a `FileUpload` object.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
JavaScript syntax:	-	*`myFileUpload`*`.select()`

This selects all the text in the `FileUpload` object so that it can be cut or copied by the user if necessary, or used as `TextRange` and have a command executed on it. It also triggers a `Select` event.

See also:	`Document.execCommand()`, `Input.select()`, `TextRange.execCommand()`

FileUpload.size (Property)

Returns the size of the file to be uploaded.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myFileUpload`*`.size`

It may be important to limit the size of files being uploaded to the server.

It is very possible that vindictive users may use file uploading to try and crash your server or deny its availability to other users by uploading massive files to it.

On the other hand, there may be limits you want to impose as a courtesy to the network managers who run the LANs and WANs where your users will be browsing. Allowing the user to upload unnecessarily large files will saturate their network as well as your server.

By testing this property, you can impose limits of acceptability on file uploads.

See also:	`Input.size`

FileUpload.type (Property)

The type of a form input element.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myFileUpload`*`.type`

The type value for a `FileUpload` is always "`file`". This value is necessary to determine the type of form element because this object is really an instance of the `Input` class.

See also:	Input.type

Property attributes:

ReadOnly.

FileUpload.value (Property)

The file name entered by the user for the file to be uploaded.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myFileUpload*.value

This property yields the name of the file that is to be uploaded. You may want to check this for its suitability. Perhaps you need a text file and the user chose an executable. Careful checking at the client can save serious problems at the server when your data processing code receives the uploaded file. Sending an image file to a server back-end that was expecting a small text file containing short lines can cause a server crash.

See also:	Input.value, UniversalFileRead

Filter (Definition)

A mechanism for providing display effects in MSIE.

The filters fall into three broad groups.

Static filters can be used (and stacked) to enhance the visual display effect of an object as it is rendered on screen. By using time-out loops you can also use them as animated effects by altering one property continuously.

The transitional filters control the change from one display to another, but can be used as static filters by freezing them at a fixed point in the transition.

The procedural filters calculate a surface according to a shading algorithm.

There are examples in the filter topics of how to use the latest variants of these filters. The older versions are now deprecated in favor of a new set and the examples have been omitted for the deprecated filters. In any case, they can all be implemented in terms of the new versions of the filters.

The major change took place at version 5.5 of MSIE and it is recommended that you use the latest filter set for any new projects.

Warnings:

- This is not the same as the event filtering provided by Netscape Navigator

See also:	`Element.filters[]`

filter – Alpha() (Filter/visual)

A visual filter for controlling transparency.

Availability:	JScript –3.0 Internet Explorer –4.0
Object properties:	`Enabled, Opacity, FinishOpacity, StartX, StartY, FinishX, FinishY, Style`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This filter is used to define a transparency level with an optional gradient effect.

The `Enabled` property can be set `true` to switch on the effect of this filter or `false` to deactivate it.

The `Opacity` property describes the initial opacity value. The values `0` and `100` represent the full range of alpha levels with `0` being fully transparent to `100` being fully opaque.

When using a gradient, the `FinishOpacity` describes the opacity required on completion.

The `StartX` and `StartY` properties are the starting coordinates of the gradient.

The `FinishX` and `FinishY` properties are the ending coordinates of the gradient.

The `Style` property defines the kind of gradient to use for the alpha channel filter. The following kinds of gradient are supported:

Index	Description
`0`	Uniform alpha level defined by the opacity name=value pair. The entire element is rendered at the same opacity level.
`1`	A linear gradient is specified. The two opacity levels define the start and end gradient points while the start and finish coordinates define the gradient normal vector.
`2`	Radial gradient starting at the `StartX`, `StartY` position and using the two opacity values for range.
`3`	Rectangular gradient spanning the rectangle defined by the start and end coordinates and using the start and end opacity values.

There are two examples. One shows the creation of the filter from script, the other implements the same effect but shows the modification of an existing object defined by `<STYLE>` tags.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as name=value pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="pulsateButton()">
<INPUT ID="MYBUTTON" TYPE="button" VALUE="Button">
<BR>
<SCRIPT>
var theOpacity = 0;
var theIncrement = 1;
function pulsateButton()
{
   theOpacity += theIncrement;

   myFilter = "Alpha(opacity="+theOpacity+")";
   document.all.MYBUTTON.style.filter = myFilter;
   if((theOpacity % 100) == 0)
   {
      theIncrement *= -1;
   }

   setTimeout("pulsateButton()", 5);
}
</SCRIPT>
</BODY>
</HTML>
-----------------------------------------------------------------
<HEAD>
<STYLE>
INPUT.aFilter {filter:Alpha(Opacity=50);}
</STYLE>
</HEAD>
<BODY onLoad="pulsateButton()">
<INPUT ID="MYBUTTON" TYPE="button" VALUE="Button" CLASS="aFilter">
<BR>
<SCRIPT>
var theOpacity = 0;
var theIncrement = 1;
var theFilter = document.all.MYBUTTON.filters[0];
function pulsateButton()
{
   theOpacity += theIncrement;

   theFilter.opacity = theOpacity;
   if((theOpacity % 100) == 0)
   {
      theIncrement *= -1;
   }

   setTimeout("pulsateButton()", 5);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, style.filter

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Opacity`	-	3.0 +	-	4.0 +	-	Warning
`FinishOpacity`	-	3.0 +	-	4.0 +	-	Warning
`StartX`	-	3.0 +	-	4.0 +	-	Warning
`StartY`	-	3.0 +	-	4.0 +	-	Warning
`FinishX`	-	3.0 +	-	4.0 +	-	Warning
`FinishY`	-	3.0 +	-	4.0 +	-	Warning
`Style`	-	3.0 +	-	4.0 +	-	Warning

filter – AlphaImageLoader() (Filter/procedural)

An image is displayed in the object with some additional control over how it is displayed.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, SizingMethod, Src`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

If you consider the content of the object and its background as two layers that are composited together, this filter interposes an additional image layer between them.

This layer can have its transparency adjusted as well as scaling and clipping to position the image where you want it.

The `Enabled` property can be set `true` to switch on the effect of this filter or `false` to deactivate it.

The `SizingMethod` property can be set to one of the following values:

- The `crop` value forces the image to be cropped to fit the HTML Element content's extent rectangle.
- The `image` value forces the HTML Element to be resized to accommodate the extent rectangle of the image file.
- The `scale` value forces the image to be scaled to fit the current extent rectangle for the HTML Element.

The `Src` property is mandatory and needs to point at a valid and absolute image URL value that can be loaded.

The example demonstrates these capabilities by cycling round all three settings using a `switch` case selector.

Example code:

```
<HTML><HEAD></HEAD>
<BODY>
<SCRIPT>
var theState = 0;
// Cycle the sizingMethod property to size the image.
function changeState(oObj)
{
   switch(theState)
   {
      case 0:
      theState = 1;
      CONTAINER.filters(0).sizingMethod = "image";
      oObj.innerText = 'Scale to fit';
      break;
      case 1:
      theState = 2;
      CONTAINER.filters(0).sizingMethod = "scale";
      oObj.innerText = 'Crop to fit';
      break;
      case 2:
      theState = 0;
      CONTAINER.filters(0).sizingMethod = "crop";
      oObj.innerText = 'Normal';
      break;
   }
}
</SCRIPT>
<DIV ID="CONTAINER" STYLE="position:absolute; left:140px; height:50; width:50;
filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(src='C:\FilterTests\Logo
150.gif', sizingMethod='scale');" >
</DIV>
<BUTTON onclick="changeState(this);">Scale to fit</BUTTON>
</BODY>
</HTML>
```

See also:	`Filter` object, Procedural surfaces, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	4.0 +	-	-
`SizingMethod`	-	5.5 +	-	5.5 +	-	-
`Src`	-	5.5 +	-	5.5 +	-	-

filter – Barn() (Filter/transition)

A transition effect with the appearance of barn doors opening or closing.

Availability:	JScript –5.5 Internet Explorer –5.5

Object properties:	Duration, Enabled, Motion, Orientation, Percent, status
Object methods:	apply(), play(), stop()

The `Duration` property controls the time it takes to play-back the transition effect.

The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.

The `Motion` property can use the values in or out to determine the direction that the transition moves in.

The `Orientation` property indicates whether the effect is applied horizontally or vertically with the values `horizontal` or `vertical`.

The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between `0` and `100`.

The `status` value can be read to determine the current disposition of the transition filter. It will return one of three values. The `0` value indicates the transition has stopped, `1` indicates that it is completed and `2` that it is still in progress.

The `apply()` method sets the transition effect to its initial condition.

The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.

The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example shows the filter being applied in a continuous time-out loop.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Barn(orientation=vertical,
motion=out) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
```

```
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC=".\Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility = "visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility = "hidden";
      DIV2.style.visibility = "visible";
   }
   else
   {
      DIV1.style.visibility = "visible";
      DIV2.style.visibility = "hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 2000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also: filter – `Iris()`, filter – `RandomBars()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Motion`	-	5.5 +	-	5.5 +	-	-
`Orientation`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – BasicImage() (Filter/visual)

Controls over the basic image display attributes of the containing HTML Element object.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, GrayScale, Invert, Mask, MaskColor, Mirror, Opacity, Rotation, XRay`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESET, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This filter was added for version 5.5 of MSIE and consolidates the image handling capabilities of the filters in earlier versions of the MSIE browser.

The `Enabled` property turns the filter on and off by assigning the `true` or `false` value to it.

The `GrayScale` property replaces the functionality of the `GrayScale()` filter. Setting it to `1` discards the color information while zero makes it inactive.

The `Invert` property replaces the functionality of the `Invert()` filter. However, note that it inverts the value in the RGB color space and not the HSV color space that the deprecated `Invert()` filter used to use.

The `Mask` property replaces the functionality of the now deprecated `Mask()` filter. Any pixels set to transparent will be replaced by the color defined in the `MaskColor` operator. This means that `MaskColor` must be specified when the `Mask` operator is used.

The `Mirror` property replaces the deprecated `FlipH()` filter. The value `0` and `1` control whether the mirror effect is applied. To reproduce the effect of the `FlipV()` filter, use the `Rotation` operator first to rotate the element by 90 degrees.

The `Opacity` property can accept a floating point value between `0.0` and `1.0`. `0.0` is completely transparent while `1.0` is completely opaque.

The `Rotation` property takes a set of values to specify which of of the four cardinal directions to rotate the object. The value `0` corresponds to no rotation. Angles of 90, 180 and 270 degrees are indicated by the values `1`,`2` and `3` respectively. This does not provide a free rotate facility. You can use it in combination with the mirroring to perform `Mirror` flips other than a straightforward left to right mirror.

The `XRay` property displays the image taking the average of the red and green values of each pixel and then inverting it. This is not quite the same as an invert on the GrayScale value. It is also different to what the earlier `XRay()` filter provided according to the documentation. Earlier `XRay()` filters were documented as proving just an edge detected outline.

The example shows an image in its normal and XRay modes.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
X-Ray--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.BasicImage(xray=1)";

document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	color value, filter – `FlipH()`, filter – `FlipV()`, filter – `Grayscale()`, filter – `Invert()`, filter – `Mask()`, filter – `Matrix()`, filter – `XRay()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-
`GrayScale`	-	5.5 +	-	5.5 +	-	-
`Invert`	-	5.5 +	-	5.5 +	-	-
`Mask`	-	5.5 +	-	5.5 +	-	-
`MaskColor`	-	5.5 +	-	5.5 +	-	-
`Mirror`	-	5.5 +	-	5.5 +	-	-
`Opacity`	-	5.5 +	-	5.5 +	-	-
`Rotation`	-	5.5 +	-	5.5 +	-	-
`XRay`	-	5.5 +	-	5.5 +	-	-

filter – BlendTrans() (Filter/blend)

A blend filter for controlling transitions.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated

This blend transition filter is used to control the fading in and out of the filtered element. This would be used to create a dissolve effect.

The duration name=value pair specifies a floating point value in seconds which controls how quickly the effect should take place.

The use of this feature is deprecated in favor of the named transitions introduced in later versions of MSIE and which can be used as static or transitional effects.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as name=value pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

See also:	filter – `RevealTrans()`, `Filter` object, `style.filter`, Transition

filter – Blinds() (Filter/transition)

A transition effect with the appearance of venetian blinds opening or closing.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`bands, Direction, Duration, Enabled, Percent, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- The `bands` property defines the number of strips that the blinds filter will use for its transition effect. The value should be in the range 1 to 100.
- The `Direction` property determines which direction the transition should proceed. It accepts the values: left, right, up or down.
- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example runs continuously in a timeout loop.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Blinds(orientation=vertical,
motion=out) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 2000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `CheckerBoard()`, filter – `Slide()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
bands	-	5.5 +	-	5.5 +	-	-
Direction	-	5.5 +	-	5.5 +	-	-
Duration	-	5.5 +	-	5.5 +	-	-
Enabled	-	5.5 +	-	5.5 +	-	-
Percent	-	5.5 +	-	5.5 +	-	-
status	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
apply()	-	5.5 +	-	5.5 +	-	-
play()	-	5.5 +	-	5.5 +	-	-
stop()	-	5.5 +	-	5.5 +	-	-

filter – Blur() (Filter/visual)

A visual filter for blurring objects.

Availability:	JScript –3.0 Internet Explorer –4.0
Object properties:	Enabled, MakeShadow, PixelRadius, ShadowOpacity
Supported by objects:	A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMEST, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP

This visual filter provides a way of adding motion blur to elements as they are drawn into the display. Versions of the MSIE browser prior to version 5.5 implement the `MotionBlur()` filter as a `Blur()` filter.

As of the IE 5.5 browser, the `Blur` effect provides a simple Gaussian blur which is non directional. The Previous motion blurring artifacts is now available with a specialized `MotionBlur()` filter.

From version 5.5 of MSIE, the properties supported by this filter have changed to this set:

- The `Enabled` property turns the blurring effect on and off when set to `true` or `false`, respectively.
- The `MakeShadow` property allows you to select between displaying the object as a shadow, or as its normal RGB values. Set to `true` to get a blurred shadow effect.
- The `PixelRadius` property takes a floating point value to indicate the radius of the blurring effect. The value ranges from 1.0 to 100.0 in pixels.
- The `ShadowOpacity` property takes a floating point value in the range 0.0 to 1.0 to indicate the opacity of the blurred image. The value 0.0 is completely transparent while 1.0 is completely opaque.

The example shows how to make a blurred shadow effect.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Blur(makeshadow=true,
pixelradius=5.0, shadowopacity=0.5)";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `MotionBlur()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-
`MakeShadow`	-	3.0 +	-	4.0 +	-	Warning
`PixelRadius`	-	3.0 +	-	4.0 +	-	Warning
`ShadowOpacity`	-	3.0 +	-	4.0 +	-	Warning

filter – CheckerBoard() (Filter/transition)

A transition effect with the appearance of chequer board blinds opening or closing.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Direction, Duration, Enabled, SquaresX, SquaresY`

This transition effect supports the following properties:

- The `Direction` property determines which direction the transition should proceed. It accepts the values: left, right, up or down.
- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.

- The `SquaresX` and `SquaresY` properties describe how many squares the effect should use across the content being transitioned.

The example runs in a continuous loop.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.CheckerBoard(direction=left,
squaresx=10, squaresy=10) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 2000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – Blinds(), Filter object, style.filter

Property	JavaScript	JScript	N	IE	Opera	Notes
Direction	-	5.5 +	-	5.5 +	-	-
Duration	-	5.5 +	-	5.5 +	-	-
Enabled	-	5.5+	-	5.5 +	-	-
SquaresX	-	5.5 +	-	5.5 +	-	-
SquaresY	-	5.5 +	-	5.5 +	-	-

filter – Chroma() (Filter/visual)

A visual filter for chroma key effects.

Availability:	JScript –3.0 Internet Explorer –4.0
Object properties:	Enabled, Color
Supported by objects:	A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP

This visual filter provides a way to define a particular color as being transparent. This is sometimes called chroma keying (a technique much used in the television industry).

The Color property defines a hex triplet value which is deemed to be transparent for this element.

Version 5.5 of the MSIE browser adds the Enabled property which controls whether the effect is applied by means of the true and false value.

The example makes black the transparent color for the image.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as name=value pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.
- This can display some unattractive visual artifacts with images that have been compressed or dithered to reduce the number of colors available. It also does not work very well round the edges of anti-aliased images.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Chroma(color=#000000)";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	Nav	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Color`	-	3.0 +	-	4.0 +	-	Warning

filter – Compositor() (Filter/visual)

As content is added to an object, it can be colored to indicate that it is changed content.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, Function`
Object methods:	`apply(), play()`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESET, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

Although this is implemented as a transitional effect, it actually only affects the display of an HTML Element in a static manner.

The Function property provides a way to select a rule for displaying a pixel color based on the two corresponding pixels in the previous and new image content of the HTML element being filtered.

The actual function is selected by a numeric value as follows (as described by the MSDN scripting reference information):

Code	Operation	Description
0	CLEAR	Neither pixel value displayed.
1	MIN(A, B)	Show the less bright of the two pixels.
2	MAX(A, B)	Show the brighter of the two pixels.
3	A	*Input A* shown, *Input B* ignored.
4	A OVER B	*Input A* displayed over *Input B*. All of *Input A* is visible, and *Input B* shows through translucent regions of *Input A*.
5	A IN B	Display all parts of *Input A* that are contained in *Input B*. Only regions with nonzero alpha values for both images are visible, and no part of *Input B* shows through.
6	A OUT B	Display all parts of *Input A* that are not contained in *Input B*. No part of *Input B* is displayed.
7	A ATOP B	Display *Input A* covering *Input B*, with each sample scaled by the alpha channel of *Input B*.
8	A SUBTRACT B	Display *Input A* with the sample color values of *Input B* subtracted from the corresponding sample color values of *Input A*. The resulting color is scaled by the alpha values of *Input A*.
9	A ADD B	Display *Input A* with the sample color values of *Input B* added to the corresponding sample color values of *Input A*. The resulting color value is scaled by the alpha value of *Input A*.
10	A XOR B	Display pixels of each set of input where the two images do not overlap. Pixels that overlap are scaled by their inverse alpha value.
19	B	*Input B* shown, *Input A* ignored.
20	B OVER A	*Input B* displayed over *Input A*. All of *Input B* is visible, and *Input A* shows through translucent regions of *Input B*.
21	B IN A	Display all parts of *Input B* that are contained in *Input A*. Only regions with nonzero alpha values for both images are visible, and no part of *Input A* shows through.
22	B OUT A	Display all parts of *Input B* that are not contained in *Input A*. No part of *Input A* is displayed.
23	B ATOP A	Display *Input B* over *Input A*, with each sample scaled by the alpha channel of *Input A*.
24	B SUBTRACT A	Display *Input B* with the sample color values of *Input A* subtracted from the corresponding sample color values of *Input B*. The resulting color is scaled by the alpha values of *Input B*.
25	B ADD A	Display *Input B* with the sample color values of *Input A* added to the corresponding sample color values of *Input B*. The resulting color value is scaled by the alpha value of *Input B*.

The `apply()` method should be executed on the `filter` object to capture the initial (*state A*) image pixel map. Then the `play()` method can be applied after some changes have taken place. At this point the *state B* image can be determined and compared with the *state A* image.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY bgcolor=gray onload="loader()">
<BUTTON onClick="filterThing()">Click me</BUTTON>
<BR>
<DIV ID="CONTAINER"
STYLE="filter:progid:DXImageTransform.Microsoft.Compositor(function=20);
position:absolute; height:300; width:300">
<IMG SRC="./Logo150.gif" STYLE="position:absolute; left:50; top:50;">
</DIV>
<SCRIPT>
function loader()
{
   CONTAINER.filters.item(0).Apply();
   CONTAINER.innerHTML = HIDDEN.innerHTML;
   CONTAINER.filters.item(0).Play();
}
function filterThing()
{
   CONTAINER.filters.item('DXImageTransform.Microsoft.Compositor').Function = 2;
}
</SCRIPT>
<DIV ID="HIDDEN" STYLE="display:none">
<IMG SRC="./Logo150.gif" STYLE="position:absolute; left:50; top:50;">
<IMG SRC="./Logo150.gif" STYLE="position:absolute; left:70; top:70;">
</DIV>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Function`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-

filter – DropShadow() (Filter/visual)

A visual filter for creating drop shadows.

Availability:	JScript –3.0 Internet Explorer –4.0

Object properties:	Enabled, OffX, OffY, Positive , Color
Supported by objects:	A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP

This visual filter creates drop shadow effects. These are quite useful for lifting control elements out of the page to make them more readily visible.

The following properties are supported:

- The `Enabled` property was added at version 5.5 of MSIE to implement a consistent way of enabling/disabling filters by means of its `true` or `false` setting.
- The `offx` and `offy` properties define the magnitude and direction of the dropshadow.
- The `positive` property specifies whether all pixels or only visible pixels generate a drop shadow. Setting this value to 0 applies a shadow based on every pixel in the element. A value of 1 only shadows non-transparent pixels.
- The `Color` property provides a way to determine the drop shadow color. This is not available on earlier versions of the MSIE browser.

The example shows the effect of adding a drop shadow to an image. In this example, you will see that the shadow has a hard edge created by a point source of light –you may want to use the `Blur` filter to provide a more natural effect with soft edges.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.DropShadow(offx=10, offy=10,
positive=1, color=gray)";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	color value, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-
`OffX`	-	3.0 +	-	4.0 +	-	Warning
`OffY`	-	3.0 +	-	4.0 +	-	Warning
`Positive`	-	3.0 +	-	4.0 +	-	Warning
`Color`	-	3.0 +	-	4.0 +	-	Warning

filter – Emboss() (Filter/visual)

Displays the image content of the HTML element as if it were an embossed effect.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, Bias`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESET, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

The grayscale values of the image are used as a height map, and an image is generated by using this height map with a light source to cast a shadow. The result is a grayscale image.

The following properties are supported:

- The `Enabled` property can be set to `true` or `false` to switch the filter on or off.
- The `Bias` property controls an intensity level that is added to the brightness of the embossed filter output. The value can range from -1.0 to 1.0 with brighter images as the value becomes more positive.

The example shows the application of an Emboss filter to an image. However when running tests, the `Bias` value did not seem to have any visible effect on MSIE 5.5 running on Windows 98.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
```

```
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Emboss()";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `Engrave()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Bias`	-	5.5 +	-	5.5 +	-	-

filter – Engrave() (Filter/visual)

An effect that is the opposite of the embossed image appearance.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, Bias`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESST, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This filter operates in a similar way to the `Emboss()` filter. In this case, the height information is inverted giving the effect of an engraved image.

The following properties are supported:

- The `Enabled` property can be set to `true` or `false` to switch the filter on or off.
- The `Bias` property controls an intensity level that is added to the brightness of the engraved filter output. The value can range from -1.0 to 1.0 with brighter images as the value becomes more positive.

The example shows the application of an `Engrave` filter to an image. However when running tests, the Bias value did not seem to have any visible effect on MSIE 5.5 running on Windows 98.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Engrave()";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `Emboss()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Bias`	-	5.5 +	-	5.5 +	-	-

filter – Fade() (Filter/transition)

A transition effect with the appearance of a dissolve between two images.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, Overlap, Percent, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `Overlap` property is a floating point value from 0.0 to 1.0 that determines what proportion of the duration time both images should be partially visible.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example runs in a continuous loop.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Fade(duration=5, overlap=2)
">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
```

```
    }
    CONTAINER.filters[0].Play();
    setTimeout("switchState()", 2000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Overlap`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – FlipH() (Filter/visual)

A visual filter for horizontal mirror effects.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated

This visual filter is used for creating symmetrically mirrored copies of an element flipped on the horizontal axis. There are no properties to use with it.

The use of this filter is now deprecated in favor of the `BasicImage()` filter that was implemented with the IE 5.5 browser. You can also use the `Matrix()` filter in place of this.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

See also:	filter – `BasicImage()`, filter – `Matrix()`, `Filter` object, `style.filter`

filter – FlipV() (Filter/visual)

A visual filter for vertical mirror effects.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated

This visual filter is used for creating symmetrically mirrored copies of an element flipped on the vertical axis. There are no properties to use with it.

The use of this filter is now deprecated in favor of the `BasicImage()` filter that was implemented with the IE 5.5 browser. You can also use the `Matrix()` filter in place of this.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

See also:	filter – `BasicImage()`, filter – `Matrix()`, `Filter` object, `style.filter`

filter – Glow() (Filter/visual)

A visual filter for adding a glow effect.

Availability:	JScript –3.0 Internet Explorer –4.0
Object properties:	`Enabled, Color, Strength`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This visual filter adds a glow effect surrounding the filtered element.

The following properties are supported:

- The `Enabled` property was added for version 5.5 of MSIE to provide a consistent interface for controlling whether a filter is active. It takes the values `true` or `false` to enable or disable the filter.
- The color property defines the color value for the glow.
- The strength property sets the intensity of the glow with a value from 1 to 255.

The glow effect is demonstrated in the example.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Glow(color=lightgreen,
strength=20)";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Color`	-	3.0 +	-	4.0 +	-	Warning
`Strength`	-	3.0 +	-	4.0 +	-	Warning

filter – Gradient() (Filter/procedural)

A procedural definition of a gradient effect.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, StartColor, StartColorStr, EndColor, EndColorStr, GradientType`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESET, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

The following properties can be used with this filter:

- The `Enabled` property can be set to `true` or `false` to control whether the `Gradient()` is used or not.
- The `StartColor` and `EndColor` properties can be specified as a 32 bit integer in decimal or hexadecimal notation. The extra 8 bits of information specify an alpha channel blending effect.

The `StartColorStr` and `EndColorStr` properties can be specified as a hexadecimal color value string including the alpha channel value. Color names may not work properly.

You should use `StartColor` or `StartColorStr` and `EndColor` or `EndColorStr` according how you want to specify the start and end colors. This is another example of where the MSIE browser syntax breaks with the traditional color naming conventions and goes off in another direction. This lack of consistency even within a browser makes it more difficult for beginners to learn how use these features.

- The `GradientType` property selects the kind of gradient to apply. It simply controls the orientation of the gradient. A zero value specifies a vertical gradient while the 1 value specifies a horizontal gradient.

The example shows how to enable the gradient and select its direction interactively.

Example code:

```
<HTML><HEAD></HEAD>
<BODY>
<SCRIPT>
var theState = 0;
var theGradientType = 0;
function selectGradientType()
{
   theGradientType = (theGradientType + 1) % 2;
   CONTAINER.filters(0).GradientType = theGradientType;
}

function switchState(anObject)
{
   switch(theState)
   {
      case 0:
         theState = 1;
         CONTAINER.filters(0).enabled = "true";
         anObject.innerText = 'Remove effect';
         break;
      case 1:
         theState = 0;
         CONTAINER.filters(0).enabled = "false";
         anObject.innerText = 'Apply effect';
         break;
   }
}
</SCRIPT>
<BUTTON onclick="switchState(this);">Apply effect</BUTTON>
<BR>
<BUTTON onclick="selectGradientType();">Next style</BUTTON>
<BR>
<DIV ID="CONTAINER" STYLE="position:absolute; left:140px; height:250; width:250;
filter:progid:DXImageTransform.Microsoft.Gradient(enabled='false',
startColorStr='#FF0000FF', endColorStr='#00FFFF00', GradientType=0);">
<CENTER><BR><BR><BR>
```

```
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</BODY>
</HTML>
```

See also:	color value, `Filter` object, Procedural surfaces, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	5.5 +	-	-
`StartColor`	-	5.5 +	-	5.5 +	-	-
`StartColorStr`	-	5.5 +	-	5.5 +	-	-
`EndColor`	-	5.5 +	-	5.5 +	-	-
`EndColorStr`	-	5.5 +	-	5.5 +	-	-
`GradientType`	-	5.5 +	-	5.5 +	-	-

filter – GradientWipe() (Filter/transition)

A transition effect with the appearance of a wipe with a soft edge.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, GradientSize, Motion, Percent, status, WipeStyle`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `GradientSize` property indicates what proportion of the object is covered by the gradient band. The value is in the range 0.0 to 1.0 and is a floating point value.
- The `Motion` property can be defined as forward or reverse to indicate the direction of the transition.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.
- The `WipeStyle` property is an integer value to determine which direction the gradient wipe travels. The value 0 selects a horizontal wipe while the value 1 selects a vertical wipe.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The play() method executes the transition effect using the control values and taking the time specified in the duration value. You can override the duration property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example runs in a continuous loop. Note that you should ensure the loop is at least long enough to cope with the effect duration otherwise a jerky transition will result.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.GradientWipe(duration=5,
gradientsize=0.4, motion=forward, wipestyle=0) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; "><CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
```

```
    }
    CONTAINER.filters[0].Play();
    setTimeout("switchState()", 7000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	5.5 +	-	5.5 +	-	-
`GradientSize`	-	5.5 +	-	5.5 +	-	-
`Motion`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-
`WipeStyle`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – Grayscale() (Filter/visual)

A visual filter for converting to a grayscale appearance.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated

This visual filter removes all color information from the filtered element and display it in grayscale mode only. There are no properties associated with it.

The use of this filter is now deprecated in favor of the `BasicImage()` filter that was implemented with the IE 5.5 browser.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

See also:	filter – `BasicImage()`, `Filter` object, `style.filter`

filter – Inset() (Filter/transition)

A diagonal wipe across the image revealing the new image.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, Percent, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

There do not appear to be any values that control the direction of the inset which wipes diagonally rightwards and downwards. The example shows how to apply this in a cyclic manner.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Inset(duration=2) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
```

```
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 7000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – Invert() (Filter/visual)

A visual filter for inverting image colors.

Availability:	JScript –3.0 Internet Explorer –4.0

This visual filter inverts the color value of every pixel in the filtered element.

There are several color models that could be used for this. The display is based around the RGB model and a printout would use the CMY or CMYK model. Inverting either of these would yield a different effect.

So, the browser will invert the colors using the HSV model. This should work consistently across all display mediums.

HSV stands for Hue, Saturation and Value (which means brightness or luminosity in most cases).

The Hue value is represented by a color wheel where the colors are specified on an angular basis from 0 to 360 degrees. Opposite sides of the wheel yield complementary colors.

The Saturation value defines the amount of color. No color at all yields a purely grayscale appearance.

The Value or lightness axis defines how bright the pixel is.

So inverting a pixel using HSV will perform these operations on discrete components of the color value:-

- Switch the color to one that is 180 degrees round the color wheel.
- Complement the saturation value. Unsaturated pixels become saturated and vice versa.
- Lightness is complemented making dark pixels light and vice versa.

Using this model to invert the pixels in an element should make it stand out clearly against a background.

There are no properties defined for this filter at present.

The use of this filter is now deprecated in favor of the `BasicImage()` filter that was implemented with the IE 5.5 browser.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

See also:	filter – `BasicImage()`, `Filter` object, `style.filter`

filter – Iris() (Filter/transition)

A transition effect with the appearance of an iris opening or closing.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, IrisStyle, Motion, Percent, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `IrisStyle` property is used to define the shape of the iris that is opened or closed. It can be one of DIAMOND, CIRCLE, CROSS, PLUS, SQUARE or STAR.
- The `Motion` property can use the values in or out to determine the direction that the transition moves in.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example shows how to call this effect using a continuous loop.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Iris(duration=2, motion=out,
irisstyle=cross) ">
```

```
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also: filter – `Barn()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	5.5 +	-	5.5 +	-	-
`IrisStyle`	-	5.5 +	-	5.5 +	-	-
`Motion`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – Light() (Filter/visual)

A visual filter for simulating a lighting model.

Availability:	JScript –3.0 Internet Explorer –4.0
Object properties:	`Enabled`
Object methods:	`clear(), addAmbient(), addCone(), addPoint(), changeColor(), changeStrength(), moveLight()`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This visual filter simulates the effect of a light source being played on the element.

The `Enabled` property is supported –it was added for version 5.5 of MSIE to provided a consistent way of controlling whether a filter was active or not.

There are also methods for defining the type of light source, the location of it, and its intensity. The light source color can also be specified. These methods are supported:

- The `clear()` method removes all currently defined light sources. It has no parameters. You should stack your light sources with this being the first one unless you want to accumulate the lighting effects during any dynamic updating.
- The `addAmbient()` method is used to add another ambient light source. The red, green, blue and strength parameter values describe its lighting properties. The parameters are presented in this order:
  - Red intensity
  - Green intensity
  - Blue intensity
  - Strength

- The `addCone()` method is used to to describe a directional light source. Its parameters include a description of the beam direction and spread. The parameters are presented in this order:
  - Left coordinate of the light source.
  - Top coordinate of the light source.
  - Z-axis level of the light source.
  - Left coordinate of the target light focus.
  - Top coordinate of the target light focus.
  - Red intensity
  - Green intensity
  - Blue intensity
  - Strength
  - Spread
- The `addPoint()` method is used to describe a point light source. The parameters are presented in this order:
  - Left coordinate of the light source.
  - Top coordinate of the light source.
  - Z-axis level of the light source.
  - Red intensity
  - Green intensity
  - Blue intensity
  - Strength
- The color of a light source can be changed by calling the `changeColor()` method. The parameters are presented in this order:
  - Light number
  - Red intensity
  - Green intensity
  - Blue intensity
  - Replace or accumulate color flag
- The intensity of the light impinging on the object can be modified for each light source with the `changeStrength()` method. The parameters are presented in this order:
  - Light number
  - Strength value
  - Replace or accumulate strength flag

- The location of a light source can be modified with the `moveLight()` method. The parameters are presented in this order:
  - Light number
  - Left coordinate of the light source.
  - Top coordinate of the light source.
  - Z-axis level of the light source.
  - Relative or absolute move flag

The example demonstrates how several light sources can be combined and then moved in a `mouseMove` event loop to simulate the effect of a torchlight in a darkened room. The `clear()` function is necessary to avoid the light sources simply accumulating a lighting effect with each mouse move. The ambient light source raises the background illumination to a dull grey.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

Example code:

```
<HTML>
<HEAD>
<STYLE>
.aFilter
{
   background-color:#FFFFFF;
   filter:light();
   position:absolute;
   top: 100;
   left: 100;
   color: cyan;
   height: 250;
   width: 250;
}
</STYLE>
</HEAD>
<BODY onMouseMove="myHandler()">
<DIV ID="EXAMPLE" CLASS="aFilter">
<BR><BR><BR>
<CENTER>
This text is highlighted<BR>
with a light source.
</CENTER>
</DIV>
<SCRIPT>
function myHandler()
{
   myLightX = 125;
   myLightY = 125;
   myLightZ = 1;
```

```
    myTargetX = event.x;
    myTargetY = event.y;
    myRed = 200;
    myGreen = 64;
    myBlue  = 32;
    myStrength = 100;
    mySpread = 220;
    EXAMPLE.filters[0].clear();
    EXAMPLE.filters[0].addAmbient(64, 100, 10, 100);
    EXAMPLE.filters[0].addCone(myLightX, myLightY, myLightZ, myTargetX, myTargetY,
myRed, myGreen, myBlue, myStrength, mySpread);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`clear()`	-	3.0 +	-	4.0 +	-	?
`addAmbient()`	-	3.0 +	-	4.0 +	-	?
`addCone()`	-	3.0 +	-	4.0 +	-	?
`addPoint()`	-	3.0 +	-	4.0 +	-	?
`changeColor()`	-	3.0 +	-	4.0 +	-	?
`changeStrength()`	-	3.0 +	-	4.0 +	-	?
`moveLight()`	-	3.0 +	-	4.0 +	-	?

filter – Mask() (Filter/visual)

A visual filter for creating a transparent mask.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated
Object properties:	`Color`

This visual filter is used for creating transparent masks. The transparent regions are defined by pixels set to the color value specified by the `Color` property.

The use of this filter is now deprecated in favor of the `BasicImage()` filter that was implemented with the IE 5.5 browser.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.
- According to the latest MSDN documentation, this filter appears to have been renamed as `MaskFilter()`.

See also:	filter – `BasicImage()`, filter – `MaskFilter()`, `Filter object`, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Color`	-	3.0 +	-	4.0 +	-	Warning, Deprecated

filter – MaskFilter() (Filter/visual)

Uses the transparent color pixels of an object into a mask.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled , Color`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This filter behaves differently according to whether a pixel in the HTML `Element` object being filtered is transparent or not.

Transparent pixels are added to the mask image. Non-transparent pixels are set to be transparent in the mask image.

The following properties are available to control this filter:

- The `Enabled` property provides a consistent means of controlling whether the filter is active or not. It can accept a value of `true` or `false`.
- The `Color` property defines the color value selected for the mask.

The example shows how a transparent GIF file can be used to cut out a shape with this filter.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.MaskFilter(color=lightgreen,
strength=20)";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	color value, filter – `Mask()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Color`	-	5.5 +	-	5.5 +	-	-

filter – Matrix() (Filter/visual)

A means of applying sophisticated rotation, translate, and scaling effects to an image using matrix transformation.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, M11, M12, M21, M22, Dx, Dy, SizingMethod, FilterType`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESET< Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

Matrix transformation of images provides the ability to do free rotation to any angle if you are prepared to do the trigonometrical maths.

This matrix transformation provides the following possibilities:

- Scaling
- Rotation
- Flipping

The matrix transformations are based on a 2x2 matrix with an additional linear vector.

The following properties are supported:

- The `Enabled` property can be set to `true` or `false` to control whether the `Matrix()` filter is active or not.
- The `M11`, `M12`, `M21` and `M22` properties contain the four values that comprise the matrix. You can define values in each one of them on its own or you can perform more complex transforms by defining several at once.
- The `Dx` and `Dy` properties take floating point values and are used to move the resulting filtered output along a linear vector. This helps to simplify the transformation matrix and allows it to be implanted as a 2x2 matrix instead of a 2x3 or 3x3 matrix.
- The `FilterType` property is used to select a method for deriving the resulting filter output. This is necessary because when transforming objects with a matrix, you need to interpolate pixel values to fill in gaps. This operator can be defined as a bilinear or nearest neighbor interpolation function. The bilinear interpolation is better at the expense of needing more compute power. The nearest neighbor interpolation technique may be more useful for animated effects.
- The `sizingMethod` property determines how the results are displayed. They can either be clipped to fit the container or the container can be resized to accommodate the new size of the transformed output. The values are "clip to original" or "auto expand".

The following simple operations can be applied:

- Flip horizontal
- Flip vertical
- Resize
- Rotate

A horizontal flip can be accomplished with this code fragment:

```
myFilter.M11 -= myFilter.M11;

myFilter.M12 -= myFilter.M12;
```

A vertical flip can be accomplished with this code fragment:

```
myFilter.M21 -= myFilter.M21;

myFilter.M22 -= myFilter.M22;
```

A resize of the whole image can be accomplished like this:

```
myFilter.M11 *= aScaleFactor;

myFilter.M12 *= aScaleFactor;

myFilter.M21 *= aScaleFactor;

myFilter.M22 *= aScaleFactor;
```

A horizontal stretch can be done like this:

```
myFilter.M11 *= aScaleFactor;

myFilter.M12 *= aScaleFactor;
```

And a vertical stretch like this:

```
myFilter.M21 *= aScaleFactor;

myFilter.M22 *= aScaleFactor;
```

A rotation is a little more complex:

```
deg2radians = Math.PI * 2 / 360;

rad = deg * deg2radians;

costheta = Math.cos(rad);

sintheta = Math.sin(rad);

myFilter.M11 = costheta;

myFilter.M12 = -sintheta;

myFilter.M21 = sintheta;

myFilter.M22 = costheta;
```

The example demonstrates these transformations individually. Strange shearing effects can be achieved by modifying the coefficients of the rotation matrix. These introduce scaling artifacts which might be useful when fitting an image into a space, perhaps for designing some VRML like projected surfaces without using VRML itself.

Example code:

```
<HTML>
<HEAD>
<STYLE>
.aFilter
{
   filter:progid:DXImageTransform.Microsoft.Matrix(sizingMethod='auto expand');
   position:absolute;
}
```

```
</STYLE>
</HEAD>
<BODY>
<TABLE BORDER=1>
<TR HEIGHT=160><TD>Normal--&gt;</TD><TD ALIGN=CENTER WIDTH=160>
<IMG ID="NORMAL" SRC="./Logo150.gif">
</TD></TR>
<TR HEIGHT=160><TD>Flip H--&gt;</TD><TD ALIGN=CENTER>
<IMG ID="FLIPH" CLASS="aFilter" SRC="./Logo150.gif">
</TD></TR>
<TR HEIGHT=160><TD>Flip V--&gt;</TD><TD>
<IMG ID="FLIPV" CLASS="aFilter" SRC="./Logo150.gif">
</TD></TR>
<TR HEIGHT=160><TD>Scale--&gt;</TD><TD ALIGN=CENTER>
<IMG ID="SCALE" CLASS="aFilter" SRC="./Logo150.gif">
</TD></TR>
<TR HEIGHT=160><TD>Scale--&gt;</TD><TD>
<IMG ID="ROTOR" CLASS="aFilter" SRC="./Logo150.gif">
</TD></TR>
</TABLE>
<SCRIPT>
FLIPH.filters.item(0).M11 *= -1;
FLIPH.filters.item(0).M12 *= -1;
FLIPH.style.top = 30+160;
FLIPH.style.left = 110;
FLIPV.filters.item(0).M21 *= -1;
FLIPV.filters.item(0).M22 *= -1;
FLIPV.style.top = 30+(160*2);
FLIPV.style.left = 110;
SCALE.filters.item(0).M11 *= 0.5;
SCALE.filters.item(0).M12 *= 0.5;
SCALE.filters.item(0).M21 *= 0.5;
SCALE.filters.item(0).M22 *= 0.5;
SCALE.style.top = 70+(160*3);
SCALE.style.left = 130;
myDeg2Rad = Math.PI*2/360;
myRadians = 30 * myDeg2Rad;
myCosine  = Math.cos(myRadians);
mySine    = Math.sin(myRadians);
ROTOR.filters.item(0).M11 = myCosine;
ROTOR.filters.item(0).M12 = -mySine;
ROTOR.filters.item(0).M21 = mySine;
ROTOR.filters.item(0).M22 = myCosine;
ROTOR.style.top = 20+(160*4);
ROTOR.style.left = 80;
</SCRIPT>
</BODY>
</HTML>
```

See also: filter – `BasicImage()`, filter – `FlipH()`, filter – `FlipV()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
Enabled	-	5.5 +	-	5.5 +	-	-
M11	-	5.5 +	-	5.5 +	-	-
M12	-	5.5 +	-	5.5 +	-	-
M21	-	5.5 +	-	5.5 +	-	-
M22	-	5.5 +	-	5.5 +	-	-
Dx	-	5.5 +	-	5.5 +	-	-
Dy	-	5.5 +	-	5.5 +	-	-
SizingMethod	-	5.5 +	-	5.5 +	-	-
FilterType	-	5.5 +	-	5.5 +	-	-

filter – MotionBlur() (Filter/visual)

An enhanced motion blur artifact that replaces the older `Blur()` filter functionality.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, Add, Direction, Strength`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This visual filter provides a way of adding motion blur to elements as they are drawn into the display.

The `Blur()` filter supplied with the version 4.0 of the MSIE browser was really a motion blur. At version 5.5 of MSIE, the filters have been enhanced and rationalized. The result is that the old `Blur()` filter has been renamed to `MotionBlur()` and a new `Blur()` filter has been introduced to provide a simpler gaussian blur effect without motion artifacts.

The following properties are supported by this filter:

- The `Enabled` property turns the motion blurring effect on and off by setting it to `true` or `false`.
- The `add` property can have two values. If it is set to 1, it includes the original image as well as the blurred image. If the value 0 is defined, just the blurred effect is displayed.
- The `direction` property defines the angle of the motion blur with respect to the original object. The value is specified in degrees from 0 to 359 moving clockwise as the values increase.
- The `strength` property defines how many pixels distance to apply the blur effect.

The example shows an image with motion blur applied.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.MotionBlur(direction=60,
strength=50, add=0)";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `Blur()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Add`	-	5.5 +	-	5.5 +	-	-
`Direction`	-	5.5 +	-	5.5 +	-	-
`Strength`	-	5.5 +	-	5.5 +	-	-

filter – Pixelate() (Filter/transition)

A transition effect with the appearance of a coarse pixelated dissolve.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, MaxSquare, Percent, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `MaxSquare` property indicates the largest possible size of a pixelated square. This can be in the range 2 to 50.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.

- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example demonstrates a gross pixelation transition effect.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Pixelate(duration=2,
maxsquare=50) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
```

```
    else
    {
       DIV1.style.visibility="visible";
       DIV2.style.visibility="hidden";
    }
    CONTAINER.filters[0].Play();
    setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	5.5 +	-	5.5 +	-	-
`MaxSquare`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – Pixelate() (Filter/visual)

An effect that simulates the pixellation achieved when lowering the display resolution of an image.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Enabled, Duration, MaxSquare, Percent, status`
Object methods:	`apply(), play(), stop()`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, FRAMESET, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

Although this may be used as a static filter, it can also be used with the `duration value`, `Apply()`, `Stop()` and `Play()` methods to control a transition effect.

The effect retains the current image size but takes groups of pixels and averages them and replaces the group with the average value. The effect is similar to that achieved by reducing the image and expanding it again without applying interpolation.

This can be useful for greeking out some content that you don't want to display. For example, you can obscure facial features or car number plates in this way.

The following properties can be applied to this filter when it is used as a static visual effect (Other transition control properties are not useful in this context):

- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `MaxSquare` property defines the maximum width in pixels of a pixelated square. This is effectively the amount of pixelation that occurs.
- The `Percent` property sets the point in the overall transition effect at which to capture the transition and use it as a static filter effect.

The example demonstrates the pixelation filter applied as a static effect.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Pixelate(maxsquare=10)";
document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	5.5 +	-	5.5 +	-	-
`Duration`	-	5.5 +	-	5.5 +	-	-
`MaxSquare`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – RadialWipe() (Filter/transition)

A transition effect with the appearance of a radar display wiping round.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, Percent, status, WipeStyle`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.
- The `WipeStyle` property is one of clock, wedge or radial which indicates the shape of the transition effect.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example demonstrates the use of the clock version of the radial wipe.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.RadialWipe(duration=2,
wipestyle=clock) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
Duration	-	5.5 +	-	5.5 +	-	-
Enabled	-	5.5 +	-	5.5 +	-	-
Percent	-	5.5 +	-	5.5 +	-	-
status	-	5.5 +	-	5.5 +	-	-
WipeStyle	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
apply()	-	5.5 +	-	5.5 +	-	-
play()	-	5.5 +	-	5.5 +	-	-
stop()	-	5.5 +	-	5.5 +	-	-

filter – RandomBars() (Filter/transition)

A transition effect with the appearance of random bars sliding down.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	Duration, Enabled, Orientation, Percent, status
Object methods:	apply(), play(), stop()

This transition effect supports the following properties:

- The `Duration` property controls the time it takes to play-back the transition effect.
- The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.
- The `Orientation` property indicates whether the effect is applied horizontally or vertically with the values `horizontal` or `vertical`.
- The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The `status` property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- The `apply()` method sets the transition effect to its initial condition.
- The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.
- The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example demonstrates the vertical orientation of this effect.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.RandomBars(duration=2,
orientation=vertical) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `Barn()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
Duration	-	5.5 +	-	5.5 +	-	-
Enabled	-	5.5 +	-	5.5 +	-	-
Orientation	-	5.5 +	-	5.5 +	-	-
Percent	-	5.5 +	-	5.5 +	-	-
status	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
apply()	-	5.5 +	-	5.5 +	-	-
play()	-	5.5 +	-	5.5 +	-	-
stop()	-	5.5 +	-	5.5 +	-	-

filter – RandomDissolve() (Filter/transition)

A transition effect with the appearance of a fine pixelated dissolve.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	Duration, Enabled, Percent, status
Object methods:	apply(), play(), stop()

This transition effect supports the following properties:

- The Duration property controls the time it takes to play-back the transition effect.
- The Enabled property provides a way to activate or inhibit the filter from working by assigning the true or false value to it.
- The Percent property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.
- The status property value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- The apply() method sets the transition effect to its initial condition.
- The play() method executes the transition effect using the control values and taking the time specified in the duration value. You can override the duration property by passing an optional duration argument to this method when it is called.
- The stop() method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an onFilterChange event handler if there is one defined.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.RandomDissolve(duration=2)
">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
Duration	-	5.5 +	-	5.5 +	-	-
Enabled	-	5.5 +	-	5.5 +	-	-
Percent	-	5.5 +	-	5.5 +	-	-
status	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
apply()	-	5.5 +	-	5.5 +	-	-
play()	-	5.5 +	-	5.5 +	-	-
stop()	-	5.5 +	-	5.5 +	-	-

filter – RevealTrans() (Filter/reveal)

A reveal filter for controlling transitions.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated

This reveal filter controls a transition effect that specifies the kind of wipe to use between the hiding and showing of a filtered element.

The duration `name=value` pair is specified in the same way as for the `blendTrans()` filter. It specifies a floating point value in seconds which controls how quickly the effect should take place.

The transition-shape `name=value` pair specifies the kind of wipe effect to use. It is an integer value which defines the following wipe patterns:

index	Description
00	Box in
01	Box out
02	Circle in
03	Circle out
04	Wipe up
05	Wipe down
06	Wipe right
07	Wipe left
08	Vertical blinds
09	Horizontal blinds
10	Checkerboard across
11	Checkerboard down
12	Random dissolve
13	Split vertical in

Table continued on following page

index	Description
14	Split vertical out
15	Split horizontal in
16	Split horizontal out
17	Strips left down
18	Strips left up
19	Strips right down
20	Strips right up
21	Random bars horizontal
22	Random bars vertical
23	Random

These are loosely modelled on some of the standard SMPTE wipes and dissolves although the full complement is not available.

This is deprecated in favor of using named filters that provide the visual transition effect.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

See also:	filter –BlendTrans(), Filter object, style.filter, Transition

filter – Shadow() (Filter/visual)

A visual filter for creating a shadow.

Availability:	JScript –3.0 Internet Explorer –4.0
Object properties:	`Enabled , Color, Direction`
Supported by objects:	`A, ACRONYM, ADDRESS, B, BDO, BIG BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP`

This visual filter is used to define a shadow effect. The filtered element is displayed as a silhouette.

The following properties are available for use with this filter:

- The `Enabled` property was added at version 5.5 of the MSIE browser to provide a consistent way of enabling and disabling filters by assigning a `true` or `false` value to it.
- The color of the cast shadow is defined by the color property and the direction property specifies the angle of the shadow with respect to the original filtered element's location. This value is specified in degrees, measured in a clockwise direction from 0 to 359 as the values increase.

You may want more control over the shadow appearance and so other filters might be closer to the desired effect. This one is good for a simple shadow effect. The example shows it being applied to an image.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Shadow(direction=120,
color=yellowgreen)";

document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`, `style.textShadow`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	Warning
`Color`	-	3.0 +	-	4.0 +	-	Warning
`Direction`	-	3.0 +	-	4.0 +	-	Warning

filter – Slide() (Filter/transition)

A transition effect with the appearance of one image sliding over another.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`bands, Duration, Enabled, Percent, SlideStyle, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- `bands`
- `Duration`
- `Enabled`
- `Percent`
- `SlideStyle`
- `Status`

The `bands` property defines the number of strips that the sliding filter will use for its transition effect.

The `Duration` property controls the time it takes to play-back the transition effect.

The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.

The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.

The `SlideStyle` property indicates what sort of sliding effect is used. It can be one of `HIDE`, `PUSH` or `SWAP`.

The `status` value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- `apply()`
- `play()`
- `stop()`

The `apply()` method sets the transition effect to its initial condition.

The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.

The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example shows one of the available variants of this filter.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Slide(duration=2, bands=5,
slidestyle=swap) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `Blinds()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`bands`	-	5.5 +	-	5.5 +	-	-
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`SlideStyle`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – Spiral() (Filter/transition)

Reveals the new image with a spiral effect.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, GridSizeX, GridSizeY, Percent, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- `Duration`
- `Enabled`
- `GridSizeX`
- `GridSizeY`
- `Percent`
- `Status`

The `Duration` property controls the time it takes to play-back the transition effect.

The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.

The `GridSizeX` and `GridSizeY` properties indicate the granularity of the effect.

The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.

The `status` value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- `apply()`
- `play()`
- `stop()`

The `apply()` method sets the transition effect to its initial condition.

The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.

The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example shows how to apply this transition.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Spiral(duration=2,
gridsizex=10, gridsizey=10) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
```

```
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `Zigzag()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	3.0 +	-	4.0 +	-	-
`GridSizeX`	-	5.5 +	-	5.5 +	-	-
`GridSizeY`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – Stretch() (Filter/transition)

A variation on a wipe effect except that the new image appears to stretch over the old one. The old one is squashed until it disappears.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, Percent, status, StretchStyle`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- `Duration`
- `Enabled`
- `Percent`
- `status`
- `StretchStyle`

The `Duration` property controls the time it takes to play-back the transition effect.

The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.

The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.

The status value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The `StretchStyle` property describes the kind of effect that the transition uses. It can be one of `HIDE`, `PUSH` or `SPIN`.

The following methods are supported by this transition filter:

- `apply()`
- `play()`
- `stop()`

The `apply()` method sets the transition effect to its initial condition.

The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.

The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

The example shows the spin variant of this effect.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Stretch(duration=2,
stretchstyle=spin) ">
```

```
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-
`StretchStyle`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
apply()	-	5.5 +	-	5.5 +	-	-
play()	-	5.5 +	-	5.5 +	-	-
stop()	-	5.5 +	-	5.5 +	-	-

filter – Strips() (Filter/transition)

Reveals new image by sliding diagonal strips across the image.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	Duration, Enabled, Motion, Percent, status
Object methods:	apply(), play(), stop()

This transition effect supports the following properties:

- Duration
- Enabled
- Motion
- Percent
- Status

The Duration property controls the time it takes to play-back the transition effect.

The Enabled property provides a way to activate or inhibit the filter from working by assigning the true or false value to it.

The Motion property indicates which order and direction the transition moves in. The value can be one of leftdown, leftup, rightdown or rightup.

The Percent property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.

The status value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- apply()
- play()
- stop()

The apply() method sets the transition effect to its initial condition.

The play() method executes the transition effect using the control values and taking the time specified in the duration value. You can override the duration property by passing an optional duration argument to this method when it is called.

The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

One variant of this filter is demonstrated in the example.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Strips(duration=2,
motion=leftdown) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
Duration	-	5.5 +	-	5.5 +	-	-
Enabled	-	3.0 +	-	4.0 +	-	-
Motion	-	5.5 +	-	5.5 +	-	-
Percent	-	5.5 +	-	5.5 +	-	-
status	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
apply()	-	5.5 +	-	5.5 +	-	-
play()	-	5.5 +	-	5.5 +	-	-
stop()	-	5.5 +	-	5.5 +	-	-

filter – Wave() (Filter/visual)

A visual filter for creating ripple effects.

Availability:	JScript –3.0 Internet Explorer –4.0
Object properties:	Enabled, Add, Freq, LightStrength, Phase, Strength
Supported by objects:	A, ACRONYM, ADDRESS, B, BDO, BIG, BLOCKQUOTE, body, BUTTON, CAPTION, CENTER, CITE, CODE, custom, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FONT, FORM, FRAME, Hn, I, IFRAME, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, MARQUEE, MENU, NOBR, OL, P, PLAINTEXT, PRE, Q, RT, RUBY, runtimeStyle, S, SAMP, SMALL, SPAN, STRIKE, STRONG, style, SUB, SUP, TABLE, TD, TEXTAREA, TH, TT, U, UL, VAR, XMP

This visual filter applies a rippled distortion effect to the filtered element. The effect is applied only along the vertical axis.

The following properties can be applied to this filter:

- Enabled
- Add
- Freq
- LightStrength
- Phase
- Strength

The Enabled property was introduced at version 5.5 of the MSIE browser to provide a consistent interface for activating and disabling filters by assigning the true or false value to it.

Like the MotionBlur() filter, the Add property specifies whether the original unmodified image should be placed over the rippled copy. A value of 1 adds the original while a value of 0 displayed only the rippled copy.

The `Freq` property specifies the number of waves to be applied as the distorting effect is rendered.

The `LightStrength` property indicates the intensity of the light playing on the rippled surface with a value in the range 0 to 100.

The `Phase` property defines the percentage offset for the sine wave curve in the range 0 to 100 which corresponds to an angular phase offset of 0 to 359 degrees.

The `Strength` property defines the wave rendering effect's intensity. This value must be between 0 and 255 with 0 providing no distortion and 255 causing a gross ripple effect.

The example shows this ripple effect being applied to an image.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Normal--&gt;
<IMG ID="NORMAL" SRC="./Logo150.gif">
Filtered--&gt;
<IMG ID="MYIMAGE" SRC="./Logo150.gif">
<BR>
<SCRIPT>
myFilter = "progid:DXImageTransform.Microsoft.Wave(freq=6, lightstrength=90,
phase=60, add=1, strength=10)";

document.all.MYIMAGE.style.filter = myFilter;
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Add`	-	3.0 +	-	4.0 +	-	Warning
`Freq`	-	3.0 +	-	4.0 +	-	Warning
`LightStrength`	-	3.0 +	-	4.0 +	-	Warning
`Phase`	-	3.0 +	-	4.0 +	-	Warning
`Strength`	-	3.0 +	-	4.0 +	-	Warning

filter – Wheel() (Filter/transition)

Reveals the new image with a rotating spoked wheel effect.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, Percent, spokes, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- `Duration`
- `Enabled`
- `Percent`
- `spokes`
- `status`

The `Duration` property controls the time it takes to play-back the transition effect.

The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.

The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.

The `spokes` property indicates how many spokes there are in the cartwheel that is used for the transition effect. The value can range from 2 to 20 with 8 being a typical value.

The `status` value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- `apply()`
- `play()`
- `stop()`

The `apply()` method sets the transition effect to its initial condition.

The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.

The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

This filter is demonstrated in the example.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="swiitchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.Wheel(duration=2, spokes=10)
">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function swiitchState()
{
CONTAINER.filters[0].Apply();
if (DIV1.style.visibility == "visible")
{
DIV1.style.visibility="hidden";
DIV2.style.visibility="visible";
}
else
{
DIV1.style.visibility="visible";
DIV2.style.visibility="hidden";
}
CONTAINER.filters[0].Play();
setTimeout("swiitchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	3.0 +	-	4.0 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`spokes`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

filter – XRay() (Filter/visual)

A visual filter that displays only the element edges.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated

This visual filter detects the visible edges of the filter element and only draws them. This might be useful for greeking a graphical object as it is dragged around with the cursor.

There are no properties for this filter.

The use of this filter is now deprecated in favor of the `BasicImage()` filter that was implemented with the version 5.5 MSIE browser.

Warnings:

- Filters are defined in style sheets as if they were a function call with its arguments expressed as `name=value` pairs. This is not the typical way to define arguments so you should be aware of this anomaly when working with filters.

See also:	filter – `BasicImage()`, `Filter` object, `style.filter`

filter – Zigzag() (Filter/transition)

Reveals the new image with a zigzag effect.

Availability:	JScript –5.5 Internet Explorer –5.5
Object properties:	`Duration, Enabled, GridSizeX, GridSizeY, Percent, status`
Object methods:	`apply(), play(), stop()`

This transition effect supports the following properties:

- `Duration`
- `Enabled`
- `GridSizeX`
- `GridSizeY`
- `Percent`
- `Status`

The `Duration` property controls the time it takes to play-back the transition effect.

The `Enabled` property provides a way to activate or inhibit the filter from working by assigning the `true` or `false` value to it.

The `GridSizeX` and `GridSizeY` properties indicate the granularity of the effect.

The `Percent` property controls the point at which the effect can be halted to provide a static effect. The value can be between 0 and 100.

The status value can be read to determine the current disposition of the transition filter. It will return one of three values. The 0 value indicates the transition has stopped, 1 indicates that it is completed and 2 that it is still in progress.

The following methods are supported by this transition filter:

- `apply()`
- `play()`
- `stop()`

The `apply()` method sets the transition effect to its initial condition.

The `play()` method executes the transition effect using the control values and taking the time specified in the duration value. You can override the `duration` property by passing an optional duration argument to this method when it is called.

The `stop()` method can be called at any time during the time the transition is running to halt the transition play-back. This will also trigger the execution of an `onFilterChange` event handler if there is one defined.

This filter is demonstrated in the example.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="switchState()">
<DIV ID="CONTAINER" STYLE="position:absolute; top: 0; left: 0; width: 300;
height:300;  filter:progid:DXImageTransform.Microsoft.ZigZag(duration=2,
gridsizex=15, gridsizey=15) ">
<DIV ID="DIV1" STYLE="position:absolute; top:50; left:10; width:240;
height:180;background:ivory">
```

```
<BR>
<BR>
<BR>
<HR>
<CENTER>
This is a DIV block containing text.
</CENTER>
<HR>
</DIV>
<DIV ID="DIV2" STYLE="visibility:hidden; position:absolute; top:50; left:10;
width:240; height:180; background:antiquewhite; ">
<CENTER>
<BR>
<IMG SRC="./Logo150.gif">
</CENTER>
</DIV>
</DIV>
<SCRIPT>
DIV1.style.visibility="visible";
function switchState()
{
   CONTAINER.filters[0].Apply();
   if (DIV1.style.visibility == "visible")
   {
      DIV1.style.visibility="hidden";
      DIV2.style.visibility="visible";
   }
   else
   {
      DIV1.style.visibility="visible";
      DIV2.style.visibility="hidden";
   }
   CONTAINER.filters[0].Play();
   setTimeout("switchState()", 5000);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	filter – `Spiral()`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Duration`	-	5.5 +	-	5.5 +	-	-
`Enabled`	-	3.0 +	-	4.0 +	-	-
`GridSizeX`	-	5.5 +	-	5.5 +	-	-
`GridSizeY`	-	5.5 +	-	5.5 +	-	-
`Percent`	-	5.5 +	-	5.5 +	-	-
`status`	-	5.5 +	-	5.5 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
`apply()`	-	5.5 +	-	5.5 +	-	-
`play()`	-	5.5 +	-	5.5 +	-	-
`stop()`	-	5.5 +	-	5.5 +	-	-

Filter object (Object/JScript)

A single `filter` object obtained from an element's filters array.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFilter* = *myElement*.filters(*anIndex*)
	IE	*myFilter* = *myElement*.filters[*anIndex*]
	IE	*myFilter* = *myFilters*[*anIndex*]
Argument list:	*anIndex*	A reference to an element in a collection
Object properties:	`enabled`	

This object defines a visual effect that is used when the display is updated as the result of a change to the content of an element. The `Filter` object has properties that relate to the individual filters. There are some common properties (and methods) that are available to all filters and in some cases the same property name can mean different things depending on the filter being applied.

You should access the `filter` objects via the *`Filters`* collection because you may have the same filter type repeated for a cascading effect and so you need to be sure that you are addressing the right one.

There are three kinds of filters that can be applied to an object.

- `Visual`
- `Reveal`
- `Blend`

A `visual` filter is used to enhance the visual appearance of objects; maybe to flip them over, add a glow effect or a drop shadow.

A `reveal` filter is used to apply a transition effect as the appearance changes.

A `blend` filter controls the speed at which a `reveal` filter is applied.

You can define more than one filter, they just need to be space separated from one another.

Here is a list of the procedural filter function names:

- `AlphaImageLoader()`
- `Gradient()`

Here is a list of the static filters supported at version 5.5 of the MSIE browser:

- `Alpha()`
- `BasicImage()`
- `Blur()`
- `Chroma()`
- `Compositor()`
- `DropShadow()`
- `Emboss()`
- `Engrave()`
- `Glow()`
- `Light()`
- `MaskFilter()`
- `Matrix()`
- `MotionBlur()`
- `Pixelate()`
- `Shadow()`
- `Wave()`

The old `blendTrans()` and `revealTrans()` filters are now replaced by these transition filters:

- `Barn()`
- `Blinds()`
- `CheckerBoard()`
- `Fade()`
- `GradientWipe()`
- `Inset()`
- `Iris()`
- `Pixelate()`
- `RadialWipe()`
- `RandomBars()`
- `RandomDissolve()`
- `Slide()`
- `Spiral()`
- `Stretch()`
- `Strips()`
- `Wheel()`
- `Zigzag()`

Filters are defined as if they were a sequence of space delimited function calls. they aren't really functions because their argument passing mechanism is not truly JavaScript based. Arguments to each filter function are defined as `name=value` pairs.

Refer to the specific topics on each filter function for details of what it does and how you can control it.

When using the filters in the context of the `style` object, the function name for each filter must be preceded by this string (although some less sophisticated filters seem to work without this):

`"progid:DXImageTransform.Microsoft."`

You can apply the filters directly as properties of the filter object that belongs to HTML `element` objects themselves.

Thus:

```
myFilter.Shadow(someAttributes)
```

Warnings:

- Filters are not supported in all versions of MSIE on the Macintosh. In fact they are not really well supported outside of the MSIE browser or the Win32 platform.
- There are various sources of documentation about these filters. There is some difference between them regarding the spelling of the filters names and the availability of the filters. The naming conventions are sometimes all lower case and at others a mixed upper and lower case. This suggests that the filter name parser may be case-insensitive. This also applies to the `name=value` pairs that are passed as arguments to the filter functions.
- Certain filter functions are no longer included in the MSDN reference material and so they may be considered to be deprecated.
- We have conformed to the case style of the MSDN reference and have included all the filters that were covered in earlier references as well as those that have been added recently. Those that appear not to be in the MSDN reference anymore are marked as deprecated as follows:
  - `FlipH()`
  - `FlipV()`
  - `Grayscale()`
  - `Invert()`
  - `Mask()`
  - `XRay()`

These are deprecated filters that used to provide blends and reveals:

- `BlendTrans()`
- `RevealTrans()`

Note that the functionality and availability of the filters has changed significantly from version 4.0 to version 5.5 of the MSIE browser and the older deprecated filters have been assimilated into the functionality of the new filter set. No previously existing filter appearance has been lost but they do need to be operated differently.

See also:	`Element.filters[]`, `onFilterChange`, `style.filter`, `style.textShadow`, Transition

Property	JavaScript	JScript	N	IE	Opera	Notes
`enabled`	-	3.0 +	-	4.0 +	-	-

Filter.enabled (Property)

A flag indicating whether a filter is enabled.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myFilter`*`.enabled`

This value can be set `true` or `false` to switch a filter in and out of the display.

Example code:

```
// Alternative ways of referring to a filter
// and controlling its enabled state.  This assumes
// the filter was instantiated with a <STYLE> tag previously.
myFilters = myElement.filters;
myFilter1 = myFilters.item(1);
myFilter2 = myFilters.item("DXImageTransform.Microsoft.Alpha");
myFilter1.enabled = true;
myFilter2.enabled = false;
```

Filters object (Object/JScript)

The `filters` object is part of the MSIE filter mechanisms and is a special case of the `collection` object.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*`myFilters`* `=` *`myElement`*`.filters`
Object properties:	`length`	
Object methods:	`item()`	

See also:	`Element.filters[]`, `Filter` object, `style.filter`

Property	JavaScript	JScript	N	IE	Opera	Notes
`length`	-	3.0 +	-	4.0 +	-	ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	Notes
`item()`	-	3.0 +	-	4.0 +	-	-

Refer to:

`Collection` object

Filters.item() (Method)

An item selector for accessing a single filter within the collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Filter object`	
JavaScript syntax:	IE	*myFilters*`.item(`*anIndex*`)`
	IE	*myFilters*`.item(`*aSelector*`)`
	IE	*myFilters*`.item(`*aSelector*`, `*anIndex*`)`
Argument list:	*anIndex*	A zero based index into the collection
	aSelector	A textual value that selects all matching objects

Refer to:

`Collection.Item()`

Filters.length (Property)

The length of the filter collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myFilters*`.length`

Property attributes:

`ReadOnly.`

Refer to:

`Collection.length`

final (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 section –7.4.3

ECMA 262 edition 3 section –7.5.3

finally ... (Statement)

Part of the `try ... catch ... finally` error-handling mechanism.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0

The ECMAScript standard (edition 2) defined the `finally` keyword and reserves it for future use. At edition 3, it was made a required keyword.

In anticipation of that, it is available in JavaScript version 1.4. This is also now supported in JScript version 5.0 as well.

Refer to the `try ... catch ... finally` topic for more details.

Warnings:

- This is not available for use server-side with Netscape Enterprise Server 3.

See also:	`catch( ... )`, throw, `try ... catch ... finally`

Cross-references:

ECMA 262 edition 2 section –7.4.3

ECMA 262 edition 3 section –7.5.2

ECMA 262 edition 3 section –12.14

find() (Method)

A means of performing a text search on the document content.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`find()`
	-	`find(`*`aSearchKey`*`)`
	-	`find(`*`aSearchKey`*`, `*`aCaseSense`*`)`
	-	`find(`*`aSearchKey`*`, `*`aCaseSense`*`, `*`aDirection`*`)`
	-	`myWindow.find()`
	-	*`myWindow`*`.find(`*`aSearchKey`*`)`
	-	*`myWindow`*`.find(`*`aSearchKey`*`, `*`aCaseSense`*`)`
	-	*`myWindow`*`.find(`*`aSearchKey`*`, `*`aCaseSense`*`, `*`aDirection`*`)`
Argument list:	*`aCaseSense`*	A switch for case sensitivity
	`aDirection`	A direction to search
	`aSearchKey`	The text to search for

Refer to:

`Window.find()`

FindProxyForURL() (Function/proxy.pac)

The main function in a `proxy.pac` file.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	`FindProxyForURL(`*`aFullURL`*`, `*`aHostname`*`)`
Argument list:	*`aFullURL`*	The complete URL to be checked for proxy access.
	`aHostname`	The hostname component of a URL for convenience

This function is called at request time for every URL. Therefore any lengthy and complex coding in here is going to be detrimental to the performance of your browser.

The `String` and `Math` object methods/functions are available in this context.

The return value from this function tells the browser how to reach the target URL.

For directly connected sites, the value "DIRECT" should be returned. This is usually appropriate for hosts within the same domain or on the same sub-net. The functions `isPlainHostName()` and `isInNet()` both return `true` for machines that should use `DIRECT` connection. However only one of them needs to return `true` for this to be the desired outcome. If both are `false`, then either a SOCKS gateway or PROXY server is needed.

For secure access using the HTTPS: protocol, the function might return "SOCKS" plus some details of the SOCKS host and port to use. You may choose to perform secure access by some other means.

For the remaining cases, your script should probably return "PROXY" with details of the PROXY server (or servers) to use. If more than one proxy server is returned, they are consulted in the order they are listed. This gives you an opportunity to add some load balancing logic if you care to. That is quite important if you have more than one proxy server, since you might return the proxies in the same order every time. That would more-or-less guarantee that the first one was loaded until it simply couldn't respond any more and the second would only kick in as a last resort. Randomizing the order in which they are presented ensures the work is fairly divided between them.

This function must return a very specific result. The following values are examples:

```
"DIRECT"

"SOCKS sockshost:1081"

"PROXY proxy1:1080 ; proxy2:1080"
```

Here we provide an example based on the one provided in the Wrox Instant JavaScript book (by Nigel MacFarlane) that illustrates all of these capabilities. The logic is unraveled to illustrate the selection technique at the expense of a minor performance hit.

Example code:

```
// Example proxy.pac file
function FindProxyForURL(aFullURL, aHostname)
{
   // Check for hosts in the same domain as the client
   if(isPlainHostName(aHostname))
   {
      return "DIRECT";
   }
   // Check for hosts in the same IP sub-net
   if(isInNet(aHostname, "192.168.1.0"))
   {
      return "DIRECT";
   }
   // Check for secure http: protocol
   if(aFullURL.substring(0, 6) == "https:")
   {
      return "SOCKS sockshost:1081";
   }
   // Check for secure news protocol
   if(aFullURL.substring(0, 6) == "snews:")
   {
      return "SOCKS sockshost:1081";
```

```
    }
    // Return a randomly selected proxy list
    if(Math.random() < 0.5)
    {
       return "PROXY proxy1:1080 ; proxy2:1080";
    }
    else
    {
       return "PROXY proxy2:1080 ; proxy1:1080";
    }
 }
```

See also:	`isInNet()`, `isPlainHostName()`, Proxies, `proxy.pac`

FlipH() (Filter/visual)

A visual filter for horizontal mirror effects.

Availability:	JScript –3.0 Internet Explorer –4.0

Refer to:

filter – `FlipH()`

FlipV() (Filter/visual)

A visual filter for vertical mirror effects.

Availability:	JScript –3.0 Internet Explorer –4.0

Refer to:

filter – `FlipV()`

float (Reserved word)

Reserved for future language enhancements.

The inclusion of this reserved keyword in the ECMAScript standard suggests that future versions of ECMAScript may be more strongly typed.

This keyword also represents a Java data type and the `float` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	`double`, Integer, `java.lang.Float`, LiveConnect, `long`, Reserved word, `short`

Cross-references:

ECMA 262 edition 2 section –7.4.3

ECMA 262 edition 3 section –7.5.3

Floating constant (Definition)

A floating constant is a literal floating point value.

A floating constant is a literal numeric value representing a floating-point number. Generally these would appear in comparison expressions, assignments or arithmetic expressions.

There may be some loss of precision with extremely long numbers or floating point values that are very small. In addition, some interpreters may perform rounding slightly differently. Fully ECMA compliant implementations should conform to the IEEE 754 specifications and therefore should be consistent with one another.

See also:	Constant, Constant expression, Floating point, IEEE 754, Number, Numeric literal

Floating point (Definition)

A type of number value.

Availability:	ECMAScript edition –2

Although a floating-point value may be equal in magnitude to an integer, logically they are different due to the typing of the value. However, typing in JavaScript is weak and a floating point value and an integer value may well compare equal where they might not have in other languages.

If ever the JavaScript language acquires a stronger typing facility, some scripts may fail on this point.

See also:	Decimal point (.), Floating constant, Floating point constant, Number

Cross-references:

ECMA 262 edition 2 section –7.7.3

ECMA 262 edition 3 section –7.8.3

O'Reilly *JavaScript Definitive Guide* –page –36

Floating point arithmetic (Definition)

Arithmetic operations performed on floating point values.

See also:	`Math` object, Floating point

Floating point constant (Definition)

A floating point value.

This is a special case of the arithmetic constant that provides non-integer values.

See also:	Arithmetic constant, Constant expression, Floating point, `Math.E`, `Math.LN10`, `Math.LN2`, `Math.LOG10E`, `Math.LOG2E`, `Math.PI`, `Math.SQRT1_2`, `Math.SQRT2`, `Number.MAX_VALUE`, `Number.MIN_VALUE`

Flow control (Definition)

Redirecting the program execution away from the linear flow.

Normal execution of the script proceeds from the top to the bottom, one statement at a time. This makes for a not very flexible or capable program.

Conditional execution of one section of code or another allows the execution flow to be controlled according to the result of a computation that yields a Boolean result. These either conditionally execute a piece of code, or not as is the case with the `if()` construct, or conditionally execute either one fragment of code or another as is the case with the `if()else` construct. The ternary operator `?:` provides an alternative way to construct an `if()else` mechanism at the expense of readability and potential obfuscation of your script.

Iterators are used to cycle round a section of code repeatedly until some condition is met.

A `for()` iterator will normally be used to index a counter to enumerate through a set of items, possibly in an array.

A `while`() iterator will execute the same code over and over until something in that code block changes the value of the condition at the head of the `while()` block.

Reserved words (as of ECMA edition 2) suggest that `do()` iterators and `switch()case` trees will be introduced later, or may be implemented now by forward-looking JavaScript interpreter builders. Edition 3 of the standard introduces these and a mandatory requirement.

See also:	Conditionally execute (?:), `do ... while( ... )`, `else ...`, `for( ... ) ...`, `for( ... in ... ) ...`, `if( ... ) ...`, `if( ... ) ... else ...`, Obfuscation, `switch( ... ) ... case: ... default: ...`, `while( ... ) ...`

Cross-references:

Wrox *Instant JavaScript* –page –22

focus() (Method)

Direct the keyboard and mouse focus onto the window.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`focus()`
	-	*`myWindow`*`.focus()`

See also:	`Input.blur(),Window.blur(),Window.focus()`

Folder object (Object/JScript)

A special JScript `folder` object.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*`myFolder`* `=` *`myFile`*`.ParentFolder`
	IE	*`myFolder`* `=` *`myFolder`*`.ParentFolder`
	IE	*`myFolder`* `=` *`myFileSystem`*`.CreateFolder(`*`aPath`*`)`
	IE	*`myFolder`* `=` *`myFileSystem`*`.GetFolder(`*`aPath`*`)`
	IE	*`myFolder`* `=` *`myFileSystem`*`.GetSpecialFolder(`*`aNumber`*`)`
	IE	*`myFolder`* `=` *`myFolders`*`.Item(`*`aPath`*`)`
Argument list:	*`aPath`*	The pathname of the folder to be created
	`aNumber`	A code referring to a special folder
Object properties:	`Attributes, DateCreated, DateLastAccessed, DateLastModified, Drive, IsRootFolder, Name, ParentFolder, Path, ShortName, ShortPath, Size, SubFolders, Type`	
Object methods:	`Copy(), Delete(), Move()`	
Collections:	`Files[], SubFolders[]`	

This is an object used in the Windows environment, probably as part of the WSH implementation to encapsulate a single folder within the file-system.

See also:	`File` object, `File.ParentFolder`, `FileSystem.CreateFolder()`, `FileSystem.GetFolder()`, `FileSystem.GetSpecialFolder()`, `Folder.ParentFolder`, `Folders.Add()`, `Folders.Item()`

Property	JavaScript	JScript	N	IE	Opera	Notes
`Attributes`	-	3.0 +	-	4.0 +	-	-
`DateCreated`	-	3.0 +	-	4.0 +	-	-
`DateLastAccessed`	-	3.0 +	-	4.0 +	-	-
`DateLastModified`	-	3.0 +	-	4.0 +	-	-
`Drive`	-	3.0 +	-	4.0 +	-	-
`IsRootFolder`	-	3.0 +	-	4.0 +	-	-
`Name`	-	3.0 +	-	4.0 +	-	-
`ParentFolder`	-	3.0 +	-	4.0 +	-	-
`Path`	-	3.0 +	-	4.0 +	-	-
`ShortName`	-	3.0 +	-	4.0 +	-	-
`ShortPath`	-	3.0 +	-	4.0 +	-	-
`Size`	-	3.0 +	-	4.0 +	-	-
`SubFolders`	-	3.0 +	-	4.0 +	-	-
`Type`	-	3.0 +	-	4.0 +	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	Notes
`Copy()`	-	3.0 +	-	4.0 +	-	-
`Delete()`	-	3.0 +	-	4.0 +	-	-
`Move()`	-	3.0 +	-	4.0 +	-	-

Folder.Attributes (Property)

The file system attributes of a folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myFolder`*`.Attributes`

Folders have attributes that control how they can be used and accessed. This property contains the attributes for the folder encapsulated by this object.

The attributes can be manipulated in a similar way to those belonging to the `File` object.

This property manages the attributes as a bit-mask with individual bits controlling different attributes. The bits can be accessed individually using the integer value corresponding to a power of 2. The table lists the integers that represent each different attribute:

You can use Bitwise `OR` expressions to merge them or accomplish the same with integer additions.

Value	Attribute
0	No special attributes –normal folder.
1	Read-only access.
2	Hidden folder.
4	Indicates a system folder.
8	Refers to drive volume label and cannot be altered.
16	Refers to a directory and cannot be changed.
32	Folder has changed and needs to be backed up again.
64	Folder object represents a shortcut and not a real folder.
128	Folder is compressed.

You cannot alter the settings of bits 8, 16, 64 and 128 as these affect the structure of a file. That is to say, you cannot change a folder into a file or disk volume.

You should read the current attributes setting and then modify it to write it back. The example illustrates some simple functions that encapsulate this conveniently.

Where the bit needs to be set, a simple bitwise `OR` with a single bit value is accomplished in a single line. To clear a bit, we could use a bitwise `AND` having the corresponding bit clear. In these examples a different technique is used for illustration where the bit is set regardless of its previous state and is then cleared using a subtraction. That saves the computation of a complex bit mask. An intermediate temporary variable is used to avoid signalling the operating system with modification unnecessarily requests.

There are other alternative ways to accomplish this and you could write some generic functions to examine, set or clear a bit in a bit-mask and then call them from each of these wrappers indicating the bit you want to operate on.

Example code:

```
// Examine the read/write flag
function isReadOnly(aFolder)
{
   return Boolean(aFolder.Attributes & 1);
}
// Set the folder read only
function setReadOnly(aFolder)
{
   aFolder.Attributes |= 1;
}
// Set the folder read/write
function setWriteOnly(aFolder)
{
```

```
   var myAttributes = aFolder.Attributes |= 1;
   aFolder.Attributes = myAttributes - 1;
}
// ------------------------------------------------
// Examine the hidden flag
function isHidden(aFolder)
{
   return Boolean(aFolder.Attributes & 2);
}
// Hide the folder
function setHidden(aFolder)
{
   aFolder.Attributes |= 2;
}
// Reveal the folder
function setVisible(aFolder)
{
   var myAttributes = aFolder.Attributes |= 2;
   aFolder.Attributes = myAttributes - 2;
}
// ------------------------------------------------
// Examine the system flag
function isSystemFolder(aFolder)
{
   return Boolean(aFolder.Attributes & 4);
}
// Set file to be a system folder
function setSystem(aFolder)
{
   aFolder.Attributes |= 4;
}
// Set file to be a non system folder
function setPublic(aFolder)
{
   var myAttributes = aFolder.Attributes |= 4;
   aFolder.Attributes = myAttributes - 4;
}
// ------------------------------------------------
// Examine the drive volume flag
function isDriveVolume(aFolder)
{
   return Boolean(aFolder.Attributes & 8);
}
// ------------------------------------------------
// Examine the folder flag
function isFolder(aFolder)
{
   return Boolean(aFolder.Attributes & 16);
}
// ------------------------------------------------
// Examine the backup flag
function needsBackup(aFolder)
{
   return Boolean(aFolder.Attributes & 32);
}
```

```
// Set backup required
function setBackup(aFolder)
{
   aFolder.Attributes |= 32;
}
// Clear backup
function clearBackup(aFolder)
{
   var myAttributes = aFolder.Attributes |= 32;
   aFolder.Attributes = myAttributes - 32;
}
// ------------------------------------------------
// Examine the shortcut flag
function isShortCut(aFolder)
{
   return Boolean(aFolder.Attributes & 64);
}
// ------------------------------------------------
// Examine the compressed flag
function isCompressed(aFolder)
{
   return Boolean(aFolder.Attributes & 128);
}
```

See also:	`File.Attributes`

Folder.Copy() (Method)

A method to copy folders.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFolder*.Copy(*aTarget*)
	IE	*myFolder*.Copy(*aTarget*, *aFlag*)
Argument list:	*aTarget*	A valid path within the destination file system
	aFlag	A flag to indicate whether to overwrite an existing folder

Although you can copy whole folders from the file system via the `FileSystem` object's `CopyFolder()` method, this is a more fine-grain means of copying folders.

It is highly recommended that you choose one of the several alternative techniques for copying folders and stick to it. It is far less confusing that way.

This copy method is better than using the `FileSystem.CopyFolder()` method because it's clearer in the script that we are operating on a `Folder`. The other method appears ambiguous because we are operating on something contained in the file system and not the object directly. It feels instinctively better to be operating on objects directly rather than by proxy.

See also:	`File.Copy(),FileSystem.CopyFolder()`

Folder.DateCreated (Property)

The date that the folder was created.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFolder`*`.DateCreated`

The file-system maintains a whole range of properties for its folders and files. This property exposes the date and time that the folder was first created.

See also:	`File.DateCreated`

Folder.DateLastAccessed (Property)

The date that the folder was last accessed.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFolder`*`.DateLastAccessed`

The file-system maintains a whole range of properties for its folders and files. This property exposes the date and time that the folder was last accessed by an application.

See also:	`File.DateLastAccessed`

Folder.DateLastModified (Property)

The date that the folder was last modified.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFolder`*`.DateLastModified`

The file-system maintains a whole range of properties for its folders and files. This property exposes the date and time that the folder was last modified by adding files to it or removing them from it. It isn't clear whether modifying a file will affect the modification date and time of a folder that contains it. It may depend on the file-system the folder exists within and how the application modifies the file. It may rewrite the file and delete the old one. In that case, modifying a file will alter the modification date and time of the folder it lives in.

See also:	`File.DateLastModified`

Folder.Delete() (Method)

A method to delete a folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFolder*`.Delete(`*aFlag*`)`
Argument list:	*aFlag*	A flag to indicate whether to force the deletion

You may remove folders via the `FileSystem.DeleteFolder()` method or with this method. Choose one and stick to it if you can.

See also:	`File.Delete(),FileSystem.DeleteFolder()`

Folder.Drive (Property)

The `drive` name that folder belongs to.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myFolder*`.Drive`

This returns a `Drive` name that describes the drive that the folder is currently stored on.

See also:	`Drive` object, `File.Drive`

Folder.Files[] (Collection)

Availability:	JScript –3.0 Internet Explorer –4.0
Property/method value type:	`Files object`

Folder.IsRootFolder (Property)

A flag indicating whether the folder is the root folder for the drive it lives in.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myFolder`*`.IsRootFolder`

It may be useful to detect whether a folder is a root folder. If you have built a tree-walking script to navigate round a file system hierarchy folder by folder, this property tells you when you have reached the top.

See also:	`Folder.ParentFolder`

Folder.Move() (Method)

A means of moving folders.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*`myFolder`*`.Move(`*`aTarget`*`)`
Argument list:	*`aTarget`*	A valid path within the destination file system

Moving or renaming folders is accomplished with this method or with the `FileSystem.MoveFolder()` method, whichever you prefer.

See also:	`File.Move(),FileSystem.MoveFolder()`

Folder.Name (Property)

The name of a folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myFolder.Name`

This property yields the name of the folder itself within the file-system's pathing syntax.

See also:	`File.Name`

Folder.ParentFolder (Property)

The folder containing the `folder` object.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Folder object`	
JavaScript syntax:	IE	*myFolder*`.ParentFolder`

You can use this as a means of walking up the folder hierarchy until you reach the root folder.

See also:	`File.ParentFolder`, `Folder` object, `Folder.IsRootFolder`

Folder.Path (Property)

The file-system path to the folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myFolder*`.Path`

This property yields the complete and fully qualified path to the folder within the context of its parent file-system.

See also:	`File.Path`

Folder.ShortName (Property)

The DOS 8.3 name for the folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myFolder*`.ShortName`

In the Windows environment, even though long file names are used from the user's point of view, at the lowest level within the file system some folders will still be stored under their old style DOS 8.3 format folder names. This property yields the DOS equivalent short folder name for a `Folder` object.

See also:	`File.ShortName`

Folder.ShortPath (Property)

The DOS 8.3 path to the folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFolder`*`.ShortPath`

In the Windows environment, even though long file and folder names are used from the user's point of view, at the lowest level within the file system, some folders and files will still be stored under their old style DOS 8.3 format file names. This property yields the DOS equivalent path name for a `Folder` object.

See also:	`File.ShortPath`

Folder.Size (Property)

The size of the folder contents.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myFolder`*`.Size`

The total size of all the files in the folder measured in bytes, is yielded by this property. At least this is consistent with the size property belonging to the `File` objects. It remains to be seen whether this actually yields the same value as you would get if you obtained the size properties from every file in the folder and summed them yourself. There may be invisible files and this property should include their sizes too.

See also:	`File.Size`

Folder.SubFolders[] (Collection)

A list of sub-folders within the receiving `folder` object.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Folders object`	
JavaScript syntax:	IE	*`myFolder`*`.SubFolders`

If you are walking up the folder hierarchy, you can use the `parentFolder` property.

This collection of `Folder` objects provides a way of traversing down the folder hierarchy to the leaf nodes at the deepest level.

See also:	`File.ParentFolder`

Folder.Type (Property)

The folder object type.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFolder`*`.type`

This is used as a way to determine the type of an object when `Folder` objects are mixed with `File` objects in a collection obtained from a `Folder` or `FileSystem`.

`Folder` objects should return a type value of "`File Folder`".

See also:	`File.Type`

Property attributes:

`ReadOnly`.

Folders object (Object/JScript)

A special JScript object that contains a collection of `Folder` objects.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*`myFileSystem`*`.Folders`
Object properties:	`Count`	
Object methods:	`Add(), Item()`	

You can access items in this array with the methods provided or you can create an `Enumerator` object to index through the collection one object at a time.

Property	Java Script	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
Count	-	3.0 +	-	4.0 +	-	-	-	-	-	-	ReadOnly

Method	Java Script	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
Add()	-	3.0 +	-	4.0 +	-	-	-	-	-	-	-
Item()	-	3.0 +	-	4.0 +	-	-	-	-	-	-	-

Folders.Add() (Method)

Add a folder to the collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myFolders*.Add(*aName*)
Argument list:	*aName*	The name of the new Folder object to be created

If you refer to an existing folder within the file system, an object is created to encapsulate it and that Folder object is then added to the Folders collection.

You can also create new folders by using a name that does not yet already exist within the folder that this collection is a child object of. The new folder will be created and then encapsulated as before.

See also:	Folder object

Folders.Count (Property)

Return a count of the number of Folder objects in the collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	Number primitive	
JavaScript syntax:	IE	*myFolders*.Count

You cannot modify this property to change the extent of the collection. It should however be modified as folders are created or destroyed within the file system.

Property attributes:

ReadOnly.

Folders.Item() (Method)

Locate and return a reference to a specifically named Folder.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Folder object`	
JavaScript syntax:	IE	*myFolders*`.Item(`*aName*`)`
Argument list:	*aName*	The name of the `Folder` object to be located

You can use this to retrieve a `Folder` object that encapsulates an existing folder. Like the `Add()` method, this can also be used to create new folders and to instantiate an encapsulating object before returning it.

See also:	`Folder` object

FONT object (Object/HTML)

An object that represents a `<FONT>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myFONT* `=` *myDocument*`.all.`*anElementID*
	IE	*myFONT* `=` `myDocument.all.tags("FONT")[anIndex]`
	IE	`myFONT =` *myDocument*`.all[`*aName*`]`
	-	*myFONT* `=` *myDocument*`.getElementById(`*anElementID*`)`
	-	*myFONT* `=` *myDocument*`.getElementsByName (`*aName*`)[`*anIndex*`]`
	-	*myFONT* `=` *myDocument* `.getElementsByTagName("FONT") [`*anIndex*`]`
HTML syntax:	`<FONT> ... </FONT>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`color, face, size`	

Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`

With this object, you can manipulate the appearance of text enclosed within the `<FONT>` tag and its corresponding `</FONT>` closure tag.

However, although this may be possible now, a more future-proof technique would be to modify the attributes of the `style` object associated with the block of text that is contained by this `FONT` object. Indeed, you may wish to replace the `<FONT>` tag with a `<DIV>` or `<SPAN>` tag and employ a `CLASS="..."` HTML tag attribute to attach `style` object to it.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`color`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`face`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`size`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

FONT.color (Property)

The color of text contained within a `<FONT>` block.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFONT*`.color`

The color of text affected by this `FONT` object will be defined in this property.

The color can be specified in the normal way according to the HTML color specifiers.

See also:	`BASEFONT.color`, Color names, Color value, `String.fontcolor()`

FONT.face (Property)

The font face for text contained within the `<FONT>` block.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFONT*`.face`

The font face to be used for text controlled by the `<FONT>` tag is defined by this property. It is appropriate to define a list of font faces in priority order in the normal way. The browser will use the first one it encounters that it has available.

FONT.size (Property)

The size of text contained within the `<FONT>` block.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myFONT*.size

The size of text rendered by the browser under control of this FONT object is controlled by this property. Absolute and relative sizes are supported in the normal way.

See also:	String.fontsize()

for(...) ... (Iterator)

An iterator mechanism –a loop construct.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera browser –3.0	
JavaScript syntax:	-	*aLabel*: for(*anInitializer*; *aCondition*; *aModifier*)
Argument list:	*aCondition*	An expression that yields a Boolean value
	aLabel	An optional label to name the iterator
	aModifier	Modifies the value being enumerated
	anInitializer	Assigns the starting value

A for loop is established by setting up the control construct in the header and associating a statement block to be evaluated each time the loop is iterated.

The control construct in a for loop has three semi-colon-separated expressions. They are all optional.

The first one initializes the enumerator. It can also declare and initialize a variable to be used for this purpose if one has not already been created.

The second is the condition to test for exit when the required number of loops has been iterated. The for() loop exits when this value becomes false. While it is true, the loop will continue to cycle.

The third is the incrementor (or decrementor if you prefer).

The code in the statement block can be completed early with the break, continue or return statements.

A break will cause the for() loop to drop out and execution to continue at the line following its statement block.

A continue statement will cycle to the next iteration and begin executing the statement block without executing the remaining lines in the block.

A return will exit the loop and its enclosing function and can only be used if the loop is executing inside a function block otherwise the return is meaningless.

You can understand how the `for()` loop works by considering how it can be restated as a `while()` loop.

```
for(anInitializer; aCondition; aModifier)
```

Can be recast as:

```
anInitializer;

while(aCondition)

{

someCode;

aModifier;

}
```

Although the items in the head of a `for()` iterator heading are optional, the semicolons must all be present to indicate the placement of any expressions in the heading.

There is an alternative construction called the `for ... in ...` statement which is especially useful for operating on objects. Refer to the `for ... in ...` topic for details.

At version 1.2 of JavaScript, a named continue can be used with this iterator. If the named continue is executed, control passes to the top of the named for loop. The increment expression is evaluated, then the test expression. If necessary the loop iterates once more.

Example code:

```
// Reccomended form
for(ii=0; ii<100; ii++)
{
   document.write("-");
}
// Possibly dangerous during maintenance
for(ii=0; ii<100; ii++)
document.write("-");
// Loop within a loop
for(ii=0; ii<100; ii++)
{
   for(jj=0; jj<ii; jj++)
   {
      document.write("-");
   }
   document.write("<BR>");
}
// Loop forever doing some task
for(;;)
{
   animateSpinningGraphic();
}
```

See also:	break, `continue`, `do ... while( ... )`, Flow control, `for( ... in ... ) ...`, Iteration statement, Label, Off by one errors, `while( ... ) ...`

Cross-references:

ECMA 262 edition 2 section –12.6.2

ECMA 262 edition 2 section –12.7

ECMA 262 edition 2 section –12.8

ECMA 262 edition 3 section –12.6.3

Wrox *Instant JavaScript* –page –24

for(... in ...) ... (Iterator)

An iterator mechanism –a loop construct.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –5.0 Internet Explorer –5.0 Netscape –2.0 Netscape Enterprise Server–2.0 Opera–3.0	
JavaScript syntax:	-	*aLabel*: for(*anLValue* in anObject) { *someCode* };
Argument list:	*aLabel*	An optional label to identify the loop
	anLValue	A value that can be assigned to
	anObject	An object whose properties will be cycled
	someCode	Some code that executed each time round the loop

A `for ... in` iteration statement is used to enumerate through the properties of an object.

The first item in the control construct is a container that a value can be assigned to. An `LValue` or Left-Hand Side expression in other words. The item following the in keyword, is the object whose properties are to be enumerated.

Each time round the loop, the name of an object property will be assigned to the `LValue` and it can then be used as an index to the property value in that object.

Properties that have the `DontEnum` attribute set will not be enumerated in this iteration statement.

During each iteration, the property name can be used as an array index key to extract the property value.

At version 1.2 of JavaScript, a named continue can be used with this iterator. If the named continue is executed, control passes to the top of the named for loop. The loop starts over with the next property name being assigned to the specified variable.

Warnings:

- It is often the case that some properties will not be enumerated with this mechanism. In particular, properties of host objects seem particularly prone to this problem.

Example code:

```
// Loop through the properties of an object only printing properties
// that have the string data type.
for(myProperty in myObject)
{
   if(typeof(myObject[myProperty]) == "string")
   {
      document.write(myProperty, myObject[myProperty]);
   }
}
```

See also: break, Compound statement, `continue`, `do ... while( ... )`, `DontEnumerate`, Flow control, `for( ... ) ...`, `Host` object, Iteration statement, Label, `while( ... ) ...`

Cross-references:

ECMA 262 edition 2 section –11.1.2

ECMA 262 edition 2 section –12.6.3

ECMA 262 edition 2 section –12.7

ECMA 262 edition 2 section –12.8

ECMA 262 edition 3 section –12.6.4

ECMA 262 edition 3 section –12.7

ECMA 262 edition 3 section –12.8

Wrox *Instant JavaScript* –page –34

Form (Definition)

A browser page that can be filled in and submitted with user specified values.

See also: `Document` object, `Form` object

Form element (Definition)

Those components that are used to construct a form document.

Refer to:

`Form.elements[]`

Form object (Object/HTML)

An object representing an HTML `<FORM>` tag.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0
Inherits from:	`Element object`
JavaScript syntax:	- *myForm* = *myDocument*.*aFormName* IE *myForm* = *myDocument*.all.*anElementID* IE *myForm* = *myDocument*.all.tags("FORM")[*anIndex*] IE *myForm* = *myDocument*.all[*aName*] - *myForm* = *myDocument*.forms.*aFormName* - *myForm* = *myDocument*.forms[*aFormName*] - *myForm* = *myDocument*.forms[*anIndex*] - *myForm* = *myDocument*.getElementById(*anElementID*) - *myForm* = *myDocument*.getElementsByName(*aName*)[*anIndex*] - *myForm* = *myDocument*.getElementsByTagName ("FORM")[*anIndex*] - *myForm* = *myFormArray*[*aFormName*] - *myForm* = *myFormArray*[*anIndex*] - *myForm* = *myInputObject*.form
HTML syntax:	`<FORM> ... </FORM>`
Argument list:	*aFormName* — The name of a form in the document *anIndex* — A reference to an element in a collection *aName* — An associative array reference *anElementID* — The `ID` value of an `Element` object
Object properties:	`acceptCharset, accessKey, action, elements, encoding, enctype, length, method, name, tabIndex, target`
Object methods:	`handleEvent(), reset(), submit()`
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReset, onSelectStart, onSubmit`
Collections:	`elements[]`

Each `<FORM>` tag creates an object in the `document.forms[]` array. However even if the forms are given names, those names become properties within the document object and are not assigned as element names in the `forms[]` array, because the associative naming is lacking. Access to form objects at level 0 of the DOM may need some attention when the DOM becomes level 1.

In MSIE, the form details are collected in a FORM object rather than a `Form` object. It is these object naming differences that can cause some problems with scripts, and it is possible that some implementations will actually make object class references case insensitive to avoid this problem.

Warnings:

- You can normally enumerate the properties in a `Form` object to access the Form input elements. However, if you place a `<TEXTAREA>` into a `<FORM>` then Netscape 4.7 on Macintosh will not enumerate the `Form` properties. The script halts but no error message is generated. MSIE has no problem enumerating a `<FORM>` containing a `<TEXTAREA>` tag.

See also:	`Collection` object, `Document.<form_name>`, `Document.forms[]`, `Element` object, `Element.all[]`, `Form.elements[]`, `Form.handleEvent()`, `Input.accessKey`, `Input.form`, `Legend` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`acceptCharset`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`accessKey`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`action`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`elements`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`encoding`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
`enctype`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`length`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	ReadOnly
`method`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`name`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`tabIndex`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`target`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-
`reset()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	-
`submit()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onReset	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-
onSubmit	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	-

Inheritance chain:

`Element` object, `Node` object

Form.acceptCharset (Property)

The character set that the form will be prepared to accept input from.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myForm*.acceptCharset

This becomes important if you are serving forms to different international language communities. You need to define the character set that your form is able to accept.

This property may return a single character set or a list of character sets that can be supported.

Form.action (Property)

A property that contains the submit action method for the <FORM>.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myForm*.action
HTML syntax:	<FORM ACTION="...">	

This value corresponds to the ACTION="..." tag attribute in the <FORM> tag that represents this Form object. This indicates the server side form handler method.

Warnings:

- The value you might store in this property is ignored by MSIE version 3 because it has read-only access to the property.

Form.elements.length (Property)

The number of input elements there are in the form. This is a reflection of the Form.length property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myForm*.elements.length

Form.elements.length is not really a property of the Form object but it is convenient to discuss it here since it is related to the Form.length property.

The structural model of the form and its length property is somewhat confusing because there is an elements[] collection which is really what is being measured. Somehow, the elements all being members of the Form object, means that the Form object's length value is really the length value of the elements[] collection.

The elements collection is really a reference to the Form object and this means the elements collection is polluted with a lot of associatively named member items that are not actually form elements at all.

It is possible that future browser versions will correct this and that a proper FormElementArray class is created. For legacy reasons, the Form.length parameter may need to persist.

See also:	Collection.length, Form.length, FormElement object, length

Property attributes:

ReadOnly.

Form.elements[] (Collection)

An array containing a list of elements belonging to a form.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera browser –3.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	-	*myForm*.elements
HTML syntax:	`<INPUT>`	

This property actually refers to the parent `Form` object which contains the elements array content superimposed on its existing properties.

Each item in this array represents an input control element within the form. Each input element is created with a separate `<INPUT>` tag. Any of the following element types can be present in a form object's `elements[]` array although it's unlikely you'll see them all there together:

- `Button`
- `Checkbox`
- `FileUpload`
- `Hidden`
- `Option`
- `Password`
- `Radio`
- `Reset`
- `Select`
- `Select (multiple)`
- `Submit`
- `Text`
- `Textarea`

See also:	`Button` object, `BUTTON` object, `Checkbox` object, `Collection` object, `FileUpload` object, `Form` object, `FormElement` object, `Hidden` object, `Option` object, `Password` object, `RadioButton` object, `ResetButton` object, `Select` object, `SubmitButton` object, `TEXTAREA` object, `TextCell` object

Property attributes:

`ReadOnly`.

Form.encoding (Property)

The type of encoding that the form needs to undergo during submission.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myForm*`.encoding`
HTML syntax:	`<FORM ENCTYPE="...">`

This value corresponds to the `ENCTYPE="..."` tag attribute in the `<FORM>` tag that represents this form object.

This specifies a MIME type for the form when it is submitted to a server.

Warnings:

- The value you might store in this property is ignored by MSIE version 3 because it can only read-access the property.
- This property is not supported on the WebTV platform.
- The DOM specification at level 1 specifies that this property should be called `enctype` and not `encoding`. In MSIE, the `enctype` property has a control attribute while the `encoding` property does not. This suggests that the `encoding` property is simply mapped to the `enctype` property internally.

See also:	JellyScript, MIME types

Form.enctype (Property)

An alias for the `Form.ecoding` property as specified by DOM level 1.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myForm*`.enctype`

Refer to:

`Form.encoding`

Form.handleEvent() (Method)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myFileUpload`*`.handleEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`Document` object, `Form` object, `handleEvent()`

Form.length (Property)

The number of elements in a form.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myForm`*`.length`

This may be the same as the `form.elements.length` value. Because of this, the `Form.elements` property may return a reference to the `Form` object itself in some implementations.

See also:	`Collection.length`, `Form.elements.length`, length

Property attributes:

`ReadOnly`.

Form.method (Property)

The method of submission for the `form` object.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myForm*.method
HTML syntax:	`<FORM METHOD="...">`	

This value corresponds to the `METHOD="..."` tag attribute in the `<FORM>` tag that represents this form object.

Warnings:

- The value you might store in this property is ignored by MSIE version 3 because it can only read-access the property.

Form.name (Property)

This corresponds to the `NAME` attribute of the `<FORM>` tag.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myForm*.name
HTML syntax:	`<FORM NAME="...">`	

Forms may be accessed by the `NAME` or `ID` attribute in some browsers. It is recommended practice to always use the `NAME` value.

Form.reset() (Method)

Invoke a form reset on the field content.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera–3.0	
JavaScript syntax:	-	*myForm*.reset()

Refer to:

onReset

Form.submit() (Method)

Invoke a form submit to the server.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0	
JavaScript syntax:	-	*myForm*.submit()

If your verification script is happy that the form data is correct, you can call this method to force the form data to be submitted.

Form.tabIndex (Property)

A control of where the form appears in the tabbing order of the page.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myForm*.tabIndex

This value indicates where in the tabbing sequence this object and any of its children will be placed. The tabbing order is used when filling in forms. Pressing the [tab] key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

Form.target (Property)

A target window or frame for the form response to be displayed in.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera–3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myForm*.target
HTML syntax:	`<FORM TARGET="...">`	

This value corresponds to the `TARGET="..."` tag attribute in the `<FORM>` tag that represents this form object.

Here are some example target values:

- `_parent`
- `_self`
- `_top`
- `_blank`
- Window name
- Frame name

Warnings:

- The value you might store in this property is ignored by MSIE version 3 because it can only read-access the property.

See also:	`<MAP TARGET="...">,Anchor.target,BASE.target, Location.target,Map.target`

Form verification (Definition)

A process of checking user entered form values.

You can attach a script to the `Submit` button in a form to verify the entire contents of the form as it is submitted.

You may prefer to add event handlers to the individual form elements to check them whenever they change.

This is very like the traditional data entry approach to building database loading systems.

Historically, a data entry system would carry out checks at the end of entering each field of data. It would then carry out checks on the whole record before loading it into the database. At that point some integrity checks may also come into play.

In the context of form data verification, our field end checks correspond to the scripts that run when a `Form` element changes. They can only logically check the internal integrity of that individual form element. They should be concerned with correct formatting of the value and range checking it for validity and size.

Our record end checks in the context of Form verification are done when the `Submit` button is pressed. This can check the integrity of data from field to field. That is best done at this point because if it's done at the field editing level, it needs to take account of default values which need to be considered as valid entry data. That complicates the field integrity checks. Cross-checking fields with one another should logically be done at a higher administrative level.

If all the fields correlate, the form can be submitted. If one or more needs to be corrected, a message can be presented and the offending item marked, selected or focussed for correction.

Formal Parameter List (Definition)

The arguments that are passed to a function when it is called.

Availability:	ECMAScript edition –2

The list of parameters in the function declaration. This defines the calling interface to the function.

As the formal parameter list is parsed, each item is represented by a property added to the variable object belonging to the owner of the evaluation. The attributes of the property being added depend on the type of code being evaluated. The values to be stored in each parameter's property attribute are determined by the caller.

If fewer values are provided than there are formal parameters, the remaining parameters will have the value undefined assigned to their properties in the variable object.

If more than one formal parameter shares the same name, then earlier values are overwritten by later ones in order of appearance in the Script Source Text. If an overwrite like this occurs and there are insufficient values passed by the caller, a previously defined value could be replaced by the undefined value.

See also:	`function( ... ) ...`, Script Source Text, Variable Declaration, Variable instantiation

Cross-references:

ECMA 262 edition 2 section –10.1.3

ECMA 262 edition 3 section –10.1.3

FormArray object (Object/browser)

The `FormArray` object is a collection of objects referring to the `Form` objects for the current document.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0	
JavaScript syntax:	-	*myFormArray* = *myDocument*.forms
Object properties:	`length`	
Object methods:	`item()`	

The array is constructed by taking the `<FORM>` tags in the document and building a unique `Form` object for each.

In MSIE, the `FormArray` is constructed by adding an element to the array for each `Form` object and setting its key to be the value of the `NAME="..."` HTML tag attribute. For two named `<FORM>` tags, there will be only two entries in the array.

In Netscape, the array is constructed a little differently. First, the `Form` objects are created as was the case with MSIE. Then they are added to the array and can be accessed numerically. The `FormArray.length` property is then set according to the number of `Form` objects in the array. Then, additional elements are added to the `FormArray` to correspond to the `NAME="..."` HTML tag attribute. If you have this in your document:

```
<FORM NAME="ONE">

...

<FORM NAME="TWO">

...
```

Then your `FormArray` will contain these entries:

- `0 -> Form ONE`
- `1 -> Form TWO`
- `ONE -> Form ONE`
- `TWO -> Form TWO`

However the length property will still only return the value 2.

If you make both <FORM> tags identical, with the same NAME value, like this:

```
<FORM NAME="ONE">
...

<FORM NAME="ONE">

...
```

Then, you will get this array:

- 0 -> First form
- 1 -> Second form
- ONE -> Form ONE

The length value still reports 2 but if you enumerate the array contents in a for(... in ...) loop, you get three entries now instead of four.

Regardless of how you define the forms in MSIE, it will always have the correct number of elements in the FormArray but they might have the same name. You can still access them numerically though.

Warnings:

- Although the FormArray is a collection, the MSIE 5.0 browser on the Macintosh will crash if you try to use the item() method on it.
- Be careful to avoid naming forms identically if you have more than one in a document. You still get the correct number of Form objects but accessing them via the FormArray may become problematic if you use the name attributes to locate them associatively.

See also:	Collection object, Document.forms[]

Property	Java Script	JScript	N	IE	Opera	HTML	Notes
length	1.0 +	3.0 +	2.0 +	4.0 +	-	-	Warning, ReadOnly

Method	Java Script	JScript	N	IE	Opera	HTML	Notes
item()	-	3.0 +	-	4.0 +	-	-	-

FormArray.item() (Method)

An item selector for picking items out of a collection of forms.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Form` object	
JavaScript syntax:	IE	*myFormArray*`.item(`*anIndex*`)`
	IE	*myFormArray*`.item(`*aSelector*`)`
	IE	*myFormArray*`.item(`*aSelector*`, `*anIndex*`)`
Argument list:	*AnIndex*	A zero based index into the collection
	aSelector	A textual value that selects all matching objects

Refer to:

`Collection.Item()`

FormArray.length (Property)

The number of forms in the form array.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myFormArray*`.length`

This property yields the number of forms in the document. Each form is represented by its own individual `Form` object.

The `length` property is consistent across Netscape and MSIE in that it reports the number of unique `Form` objects that are available. However, this is not the length of the array in Netscape because additional elements are placed into the array to access the `Form` objects associatively by name.

Warnings:

- The `length` value is only correct in Netscape if you are accessing the array with numeric index values. You will get more than *myFormArray*`.length` objects if you enumerate the array with a `for( ... in ... )` loop. In that case you will visit each `Form` object more than once.

See also:	`Collection.length`

Property attributes:

ReadOnly.

FormElement object (Object/browser)

An object representing an HTML `<INPUT>` tag in a `<FORM>`.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Inherits from:	Input object	
JavaScript syntax:	IE	*myFormElement* = *myDocument*.all.anElementID
	IE	*myFormElement* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*myFormElement* = *myDocument*.all[*aName*]
	-	*myFormElement* = *myDocument*.getElementById(*anElementID*)
	-	*myFormElement* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myFormElement* = *myForm*.*anElementName*
	-	*myFormElement* = *myForm*.elements[*anItemIndex*]
	-	*myFormElement* = myForm[*anIndex*]
	-	*myFormElement* = *myFormElements*Array[*anItemIndex*]
	-	*myFormElement* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
	anItemIndex	A reference to a single item within the form elements array
Event handlers:	`onAfterUpdate, onBeforeUpdate, onBlur, onChange, onClick, onDblClick, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onRowEnter, onRowExit, onSelect`	

This is a generic description of a `form element` object. The object will really be a concrete manifestation of a particular class but is available generally as an item in the elements array that belongs to the form.

Refer to the `Input` object topics for a more detailed description of `FormElement` functionality.

See also:	`Button` object, `BUTTON` object, `Checkbox` object, `Element` object, `Element.all[]`, `Form.elements.length`, `Form.elements[]`, `Input` object, `Password` object, `RadioButton` object, `ResetButton` object, `Select` object, `SubmitButton` object, `TextCell` object

Event name	JavaScript	JScript	N	IE	Opera	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	Warning
onChange	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-
onClick	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	4.0 +	Warning
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	4.0 +	Warning
onMouseOver	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	4.0 +	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-
onSelect	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

FormElementsArray object (Object/browser)

A collection containing the input elements of a form. This is provided by adding properties to the `FORM` object the elements belong to.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
JavaScript syntax:	-	*myFormElementsArray* = *myForm*.elements
Object properties:	length	

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
length	-	3.0 +	-	4.0 +	-	-	ReadOnly

Refer to:

`Collection` object

FormElementsArray.length (Property)

The number of input elements within a form.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myFormElementsArray`*`.length`

Property attributes:

`ReadOnly`.

Refer to:

`Collection.length`

forward() (Method)

Equivalent to the user clicking on the `[FORWARD]` button.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`forward()`
	-	*`myWindow`*`.forward()`
See also:	`History.forward()`	

Refer to:

`Window.forward()`

frame (Property)

This is another name for self and window.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`Window object`	
JavaScript syntax:	IE	`frame`
	IE	*`myWindow`*`.frame`

Property attributes:

`ReadOnly`.

Refer to:

`Window.frame`

Frame object (Object/DOM)

An object representing an HTML `<FRAME>` tag.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Inherits from:	`Window` object	
JavaScript syntax:	-	*myFrame* = frame
	-	*myFrame* = frames[*anIndex*]
	IE	*myFrame* = *myDocument*.all.*aFrameID*
	IE	*myFrame* = *myDocument*.all.tags("FRAME")[*anIndex*]
	IE	*myFrame* = *myDocument*.all[*aName*]
	-	*myFrame* = *myDocument*.getElementById(*anElementID*)
	-	*myFrame* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myFrame* = *myDocument*.getElementsByTagName("FRAME")[*anIndex*]
	IE	*myFrame* = *myDocument*.parentWindow
	-	*myFrame* = *myFrame*Array[*anIndex*]
	-	*myFrame* = parent
	-	*myFrame* = self
	-	*myFrame* = top
	-	*myFrame* = window
HTML syntax:	`<FRAME>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	borderColor, className, dataFld, dataSrc, defaultStatus, frameBorder, height, isTextEdit, lang, language, longDesc, marginHeight, marginWidth, name, noResize, parent, parentElement, parentTextEdit, scrolling, sourceIndex, src, style, tagName, title, top	
Object methods:	close(), contains(), getAttribute(), removeAttribute(), setAttribute()	

Event handlers:	onAfterPrint, onBeforePrint, onBeforeUnload, onBlur, onDragDrop, onError, onFocus, onHelp, onLoad, onMouseMove, onMove, onResize, onScroll, onUnload
Collections:	all[], attributes[], children[]

The methods and properties of a frame are the same as those for a window object.

Note that MSIE supports a FRAME object as opposed to a Frame object for the management of frames within a frame-set.

DOM level 2 adds the following properties:

- contentDocument

Warnings:

- Be aware that if you store a reference to a frame object and the frame is closed, if you don't dispose of the reference to the frame object then it cannot be garbage collected. A frame object with no associated frame is not much use unless you need to keep the object persistent due to having added some properties to it. If this is the case, then arguably the frame object was the wrong place to put such things.

See also:	BODY object, captureEvents(), Collection object, Document object, Document.activeElement, Document.captureEvents(), Document.frames[], Document.parentWindow, Document.releaseEvents(), Frames object, Global object, IFRAME object, Layer.captureEvents(), Layer.releaseEvents(), Layer.window, self

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
borderColor	-	3.0 +	-	4.0 +	-	-	-	-
className	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
dataFld	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
dataSrc	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
defaultStatus	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	-
frameBorder	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
height	-	3.0 +	-	4.0 +	-	-	-	Warning
isTextEdit	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
lang	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
language	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
longDesc	1.5 +	-	6.0 +	-	-	1 +	-	-
marginHeight	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
marginWidth	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
name	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
noResize	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
parent	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	-
parentElement	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
parentTextEdit	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
scrolling	1.5 +	3.0 +	2.0 +	3.0 +	-	1 +	-	Warning
sourceIndex	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
src	1.5 +	3.0 +	2.0 +	3.0 +	-	1 +	-	-
style	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
tagName	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
title	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
top	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
close()	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	Warning
contains()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
getAttribute()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
removeAttribute()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
setAttribute()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterPrint	-	5.0 +	-	5.0 +	-	-	-	-
onBeforePrint	-	5.0 +	-	5.0 +	-	-	-	-
onBeforeUnload	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onDragDrop	1.2 +	-	4.0 +	-	-	-	-	-
onError	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onLoad	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMove	1.2 +	-	4.0 +	-	-	-	-	-
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onUnload	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning

Inheritance chain:

Window object

Frame.borderColor (Property)

The color of a border around a frame.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myFrame*.borderColor

You can use this property to determine the color of a border surrounding a frame.

See also:	color names, color value

Frame.close() (Method)

A redundant method that should be ignored.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myFrame*.close()

This method is inherited from the `Window` object class but is of no use in the context of a `Frame` object.

Warnings:

- Attempting to close a frame in a frame-set serves no purpose whatsoever.

See also:	`Window.close()`

Frame.defaultStatus (Property)

A default status value that only applies when the mouse is in the frame.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFrame*.defaultStatus

Refer to:

`Window.defaultStatus`

Frame.frameBorder (Property)

A control attribute for borders around frames.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive.`	
JavaScript syntax:	-	*myFrame*`.frameBorder`

this property provides a way to switch frame borders on and off. Although it is a switching property and behaves as if it were a Boolean value it really is implemented as a String value.

This is because it accepts a yes/no value or a 0/1 value depending on which browser you are using. The following values may be used:

- yes –Supported by all browsers to turn frame borders on
- no –Supported by all browsers to turn frame borders off
- 0 –Supported by MSIE to turn frame borders off
- 1 –Supported by MSIE to turn frame borders on

See also:	`IFRAME.frameBorder, IFRAME.frameSpacing`

Frame.height (Property)

The height of a frame.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myFrame*`.height`

Frames can vary in size from browser to browser. Although you can measure the height of a frame, setting it accurately to a desired and exact size is virtually impossible. MSIE is marginally better at this than Netscape.

Warnings:

- Height and width control of frames in Netscape is nothing less than a lottery. There are all kinds of platform dependant weirdness that crop up and you may find yourself doing a browser sniff and building in corrective `document.writes()` that fix up the Frame sizes on the fly.
- Note that there is no width property to be able to measure the frame width value.

See also:	`IFRAME.height`

Frame.longDesc (Property)

This is a URL which points at a document containing a long description of the contents of this frame.

Availability:	DOM level –1 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myFrame*`.longDesc`

See also:	`IFRAME.longDesc`

Frame.marginHeight (Property)

The vertical height of a top and bottom margin round a frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFrame*`.marginHeight`

Margins flow round the entire frame. You cannot operate on them individually, but you can operate on the vertical and horizontal margins separately.

This property is the size of the margins at the top and bottom of the frame.

See also:	`IFRAME.frameSpacing`, `IFRAME.marginHeight`

Frame.marginWidth (Property)

The horizontal width of a margin round a frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFrame*.marginWidth

Margins flow round the entire frame. You cannot operate on them individually but you can operate on the vertical and horizontal margins separately.

This property is the size of the margins to the left and right of the frame.

See also:	`IFRAME.frameSpacing`, `IFRAME.marginWidth`

Frame.name (Property)

This corresponds to the `NAME` attribute of the `<FRAME>` tag.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFrame*.name
HTML syntax:	`<FRAME NAME="...">`	

See also:	IFRAME.name

Refer to:

`Window.name`

Frame.noResize (Property)

A switch to control the resize box on a frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myFrame*.noResize

This is a Boolean value to turn the frame resizing control on and off. Unlike the border control, this is a truly Boolean value accepting either `true` or `false` as its setting.

See also:	`IFRAME.noResize`

Frame.parent (Property)

A reference to the window (frame) that contains the frame.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*myFrame*.parent

See also:	`Dialog` object, `Window.parent`

Frame.scrolling (Property)

A switch to control the appearance of scrollbars on a frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –3.0 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFrame*.scrolling

Yet another deviant property in the browser mess! Having just discussed the `noResize` property which is a switching mechanism having a Boolean setting, here we are with another switching property. However, as is the case with border controls, this one is not Boolean. Instead it accepts the values:

- `yes`
- `no`
- `auto`

Warnings:

- Be aware that if you allow the scrollbars to be invisible by setting this property to "no" or "auto" when they are not present, Netscape will ignore the `scroll()` method that you apply to the `Window` object. This means that to scroll the content of a window, you must have visible scrollbars. MSIE is far more forgiving and lets you scroll as much as you like regardless of whether the scrollbars are present or not.
- You can work around this in Netscape by creating Layers that can be scrolled independently of the presence of window scrollbars.

See also:	`IFRAME.scrolling`

Frame.src (Property)

The URL for a document contained within the frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –3.0 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFrame*`.src`

Frames have their content loaded in much the same way as for a window. This property describes the source URL of the document that controls the content of a frame. You can reload the frame by redefining this value but it is probably better to use the `location.href` value for that.

See also:	`IFRAME.src`

Frame.top (Property)

The top-level window that owns the hierarchy the current frame lives in.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0
Property/method value type:	`Window object`
JavaScript syntax:	- *myFrame*`.top`

Property attributes:

`ReadOnly`.

Refer to:

`Window.top`

FrameArray object (Object/browser)

A collection of frames within a window. Actually there is no `FrameArray` object type because it is implemented as a `Frames` object.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0
JavaScript syntax:	- *myFrame*`Array = frames` - *myFrame*`Array = `*myDocument*`.frames` - *myFrame*`Array = `*myWindow*`.frames`
Object properties:	`length`
Object methods:	`item()`
See also:	`Collection` object, `Frame` object, `Window.frames[]`, `Frames` object

Property	JavaScript	JScript	N	IE	Opera	Notes
`length`	1.0 +	1.0 +	2.0 +	3.02 +	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	Notes
`item()`	-	3.0 +	-	4.0 +	-	-

FrameArray.item() (Method)

An item picker for accessing individual frames within a collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Frame object`	
JavaScript syntax:	IE	*myFrame*`Array.item(`*anIndex*`)`
	IE	*myFrame*`Array.item(`*aSelector*`)`
	IE	*myFrame*`Array.item(`*aSelector*`, `*anIndex*`)`
Argument list:	*anIndex*	A zero based index into the collection
	aSelector	A textual value that selects all matching objects

Refer to:

`Collection.Item()`

FrameArray.length (Property)

The number of frame objects in a frame collection.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`frames.length`
	-	*myDocument*`.frames.length`
	-	*myFrame*`Array.length`
	-	*myWindow*`.frames.length`

Property attributes:

`ReadOnly`.

Refer to:

`Collection.length`

frameRate (Property)

An indication of the frame rate for the current display.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	`frameRate`
	N	*`myWindow`*`.frameRate`

Property attributes:

`ReadOnly`.

Refer to:

`Window.frameRate`

Frames object (Object/browser)

A collection of frames belonging to a document or window.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
JavaScript syntax:	-	*`myFrames`* `= frames`
	-	*`myFrames`* `=` *`myDocument`*`.frames`
	-	*`myFrames`* `=` *`myWindow`*`.frames`
Object properties:	`length`	
Object methods:	`item()`	

An array of `Frame` objects contained within the document or window.

If the `Frames` object were named consistently with the rest of MSIE, it might be called a `FrameArray` object.

Note that `Frame` objects is really another name for `Window` objects.

Each entry in the `Frames` object is a reference to a `Window` object for the specified frame.

Giving the individual frames a name with the `NAME=". . ."` HTML tag attribute does not feed through into the `Frames` array. The entries are simply numbered with their index value.

The `frames` property in Netscape points at a `Window` object rather than a `Frames` object. Although there is no `Frames` array in Netscape, the `Window` object acquires a length value which is the number of `Frame` objects. However, the frames are not enumerable by index number. We can simulate the functionality of the `Frames` array but to do this, we will need to enumerate all the properties of the `Window` object and eliminate the ones we don't want.

The example below illustrates how to enumerate through the properties of a `Window` object and extract only those which are valid `Frame` objects. The `toString` function forces the new array to pose as a new object class. This actually works in MSIE and Netscape and yields an array that is consistent in both browsers.

Warnings:

- The `frames` property in MSIE points at a `Frames` array object that is easy to enumerate and operate on by itself. In Netscape, the properties that would have been stored in the `Frames` array in MSIE are simply dumped into the global property space, and stored in a `Window` object. This not only makes them harder to find but much harder to operate on in a logical way. Yet again it illustrates the need for script developers to be aware of the finer points regarding the differences between MSIE and Netscape.

Example code:

```
// Build a FrameArray in Netscape
var myIndex = 0;
var myFramesArray = new Array();
myFramesArray.toString = function () { return "[object FrameArray]"; }

for(var myProp in frames)
{
   if(isDesiredFrameObject(myProp, frames[myProp]))
   {
      myFramesArray[myIndex] = frames[myProp];
      myIndex++;
   }
}

// Select genuine frame objects
function isDesiredFrameObject(aProperty, anObject)
{
   if(toString(anObject) == "[object Window]")
   {
      switch(aProperty)
      {
         case "frames" :
         case "parent" :
         case "top"    :
         case "self"   :
            return false;
         default       :
            return true;
      }
   }
   return false;
}
```

See also:	`Collection` object, `Document.frames[]`, `Frame` object, `Window` object, `Window.frames[]`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`length`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
`item()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	Warning

Frames.length (Property)

The number of frames in the window to which the frames array belongs. The number of inline frames in the current document.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`frames.length`
	-	*`myDocument`*`.frames.length`
	-	*`myWindow`*`.frames.length`
HTML syntax:	`<IFRAME>`	

See also:	`Frames` object, `Collection.length`

Property attributes:

`ReadOnly`.

Cross-references:

Wrox *Instant JavaScript* –page –81

frames[] (Collection)

The frames array belongs to the `window` object.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Frames object`	
JavaScript syntax:	-	`frames`
	-	*`myWindow`*`.frames`
HTML syntax:	`<FRAME>`	

See also:	`Document.frames[]`, `Frames` object, `Window.frames[]`

Property attributes:

`ReadOnly`.

Cross-references:

Wrox *Instant JavaScript* –page –81

FRAMESET object (Object/HTML)

An object that represents a `<FRAMESET>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –3.0 Netscape –2.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*`myFrameSET`* `= ` *`myDocument`*`.all.`*`aFramesetID`*
	IE	*`myFrameSET`* `= ` *`myDocument`*`.all.`*`anElementID`*
	IE	*`myFrameSET`* `= ` *`myDocument`*`.all.tags("FRAMESET")[`*`anIndex`*`]`
	IE	*`myFrameSET`* `= ` *`myDocument`*`.all[`*`aName`*`]`
	-	*`myFrameSET`* `= ` *`myDocument`*`.getElementById(`*`anElementID`*`)`
	-	*`myFrameSET`* `= ` *`myDocument`*`.getElementsByName(`*`aName`*`)[`*`anIndex`*`]`
	-	*`myFrameSET`* `= ` *`myDocument`*`.getElementsByTagName` `("FRAMESET") [`*`anIndex`*`]`

HTML syntax:	`<FRAMESET> ... </FRAMESET>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`accessKey, border, borderColor, cols, frameBorder, frameSpacing, rows, tabIndex`	
Event handlers:	`onBeforeUnload, onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onUnload`	

This object encapsulates the top level frame-set in a multi-paned window. You need to be a little bit cunning to access the object for the `FRAMESET` because normally you are running a script that lives inside one of its frames.

Logically, you might assume that a `FRAMESET` object has a property with a list of frames contained within it. Actually, you can navigate the hierarchy but it's via the usual DOM properties such as `childNodes` and `firstChild` etc.

The `FRAMESET` object can be a useful way of maintaining session state. Although it is only present in the MSIE browser, the Netscape browser can accomplish the same session store techniques with the `global` object that belongs to the document containing the frame-set.

See also:	`Element` object

Property	Java Script	JScript	N	IE	Opera	DOM	HTML	Notes
`accessKey`	-	3.0 +	-	4.0 +	-	-	-	-
`border`	-	3.0 +	-	4.0 +	-	-	-	-
`borderColor`	-	3.0 +	-	4.0 +	-	-	-	-
`cols`	1.5 +	3.0 +	2.0 +	3.0 +	-	1 +	-	-
`frameBorder`	-	3.0 +	-	4.0 +	-	-	-	-
`frameSpacing`	-	3.0 +	-	4.0 +	-	-	-	-
`rows`	1.5 +	3.0 +	2.0 +	3.0 +	-	1 +	-	-
`tabIndex`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	Java Script	JScript	N	IE	Opera	DOM	HTML	Notes
`onBeforeUnload`	-	3.0 +	-	4.0 +	-	-	-	-
`onClick`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	Java Script	JScript	N	IE	Opera	DOM	HTML	Notes
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onLoad	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onUnload	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning

Inheritance chain:

`Element` object, `Node` object

FRAMESET.accessKey (Property)

A key that needs to be pressed before the input object will respond to data entry.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFrameSET`*`.accessKey`

The key defined in this property needs to be held down for any input events to be triggered on this object or its children.

FRAMESET.border (Property)

A switching attribute that controls the border around the frames in a frame-set.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFrameSET`*`.border`

This property indicates the thickness of frame borders around all the frames. The borders are turned on and off with the `frameBorder` property.

Even if the borders are turned off, the value of this property controls the spacing between frames. Setting the border color forces the borders to appear regardless of the `frameBorder` setting.

See also:	`FRAMESET.frameBorder`, `FRAMESET.frameSpacing`

FRAMESET.borderColor (Property)

An attribute controlling the color of a border around the frames in a frame-set.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myFrameSET`*`.borderColor`

You can use this property to determine the color of a border surrounding a frame. The borders are turned on and off with the `frameBorder` property.

Even if the borders are turned off, the value of the border property controls the spacing between frames. Setting the border color forces the borders to appear regardless of the `frameBorder` setting.

FRAMESET.cols (Property)

The column arrangement of frames within a frame-set.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –3.0 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myFrameSET`*`.cols`

The value defined in the `COLS` HTML tag attribute is reflected into this property. It appears exactly as it does in the HTML document source.

FRAMESET.frameBorder (Property)

A switching attribute for the border around individual frames within the frame-set.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myFrameSET`*`.frameBorder`

This property controls whether the frame borders are visible or not.

Even if the borders are turned off, the value of the border property controls the spacing between frames. Setting the border color forces the borders to appear regardless of the `frameBorder` setting.

See also:	`FRAMESET.border`

FRAMESET.frameSpacing (Property)

A property indicating the amount of space between frames within the frame-set.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myFrameSET*`.frameSpacing`

The spacing distance between frames is specified in this property. Actually, from a functional point of view, it behaves exactly the same as the border property.

See also:	`FRAMESET.border`

FRAMESET.rows (Property)

The row control attribute for the frame-set.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –3.0 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFrameSET*`.rows`

The value defined in the `ROWS` HTML tag attribute is reflected into this property. It appears exactly as it does in the HTML document source.

FRAMESET.tabIndex (Property)

A numeric ordering of the `<FRAMESET>` within the parent document's tabbing sequence.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myFrameSET*`.tabIndex`

This value indicates where in the tabbing sequence this object and any of its children will be placed. The tabbing order is used when filling in forms. Pressing the `[tab]` key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

Free-format language (Definition)

A somewhat relaxed layout strategy for source code.

JavaScript is a free-format language. That means it has a very relaxed attitude to the organization and layout of the script source text. You can place multiple statements on one line and separate them with semi-colons. It is also forgiving enough to allow you to omit semi-colons, which it will put in for you as the script is executed. There are some cases where this cannot apply because for JavaScript to do this, the programmer's original intent must be quite unambiguous. This is called automatic semi-colon insertion and is covered in a topic of its own.

See also:	Automatic semi-colon insertion, JavaScript language

Cross-references:

Wrox *Instant JavaScript* –page –17

ftp: URL (Request method)

A request from a web browser to an ftp server to send a file.

This will download a file using the FTP protocol. In most respects it is very like accessing with HTTP. However, you may need to specify a user name and password.

See also:	javascript: URL, URL

Function (Definition)

The action of executing a function object's script source text.

Availability:	ECMAScript edition –2

Functions and methods are similar but not the same thing.

Methods provide a function like behavior because they can be called and sometimes return a value. They operate on the content of the receiving object to which they are attached and yield information about the internals of the object.

Functions perform some transformation on the input arguments and return a result that is dependent on them. This computation is complete independent of the contents of any object.

Accessor methods to set and get internal values of an object are presented externally as properties.

In JavaScript, functions are first class data items. That means they are implemented as objects and can be manipulated in expressions and assigned to `LValues`. You could create an array that contains a collection of function objects. You can pass a function as a parameter to another function or store a reference to it in a variable. To do this in a compiled language would require you to do some pointer manipulation. In Objective C, this kind of functionality is called dynamic late binding and allows methods to be attached to event triggers at run-time.

A function is a construct that performs a task. It is derived from the procedural language model rather than the Object Oriented Language model. Procedural language functions are analogous to Object Oriented language methods but they are not quite the same. Functions are called and carry out some procedural task, while methods are invoked by sending a message to a receiving object – they may however belong to a class or an object instance of a class. Class methods are most likely to be factory methods for creating instance objects of that class.

Function calls can only be received by function objects otherwise a runtime error results – however, those `Function` objects generally belong to other objects and are stored as properties of the owner object.

A method is a specially written function that operates on the receiving object and can be shared between several objects using the `'this'` variable to refer to the receiving object.

Associating a function with an object renders that function the active code to be called when a method is invoked on that object. Invoking a method sends a message to the object, the receiving object looks for a property with the message name which contains a reference to callable function object. If it doesn't have one, it calls its parent and delegates the message upwards.

In JavaScript version 1.2, function definitions can be nested within one another. You can end up with local scope chains being nested within one another. Be sure to scope any variables to exist within the level of function nesting that you need. You can accomplish this with the `var` keyword.

Example code:

```
// Example of JavaScript version  1.2 function nesting
// Create a globally scoped value
var scope = "Global"

// Define a function object and store a global reference to it
myFunc1 = function outer()
{
   var scope = "Outer"
   // Define an inner anonymous function and store a global reference to it
   myFunc2 = function ()
   {
      var scope = "Inner"
      return scope;
   }
   return scope;
}

// Call the various functions to establish the scope of the variables
document.write("<TABLE BORDER=1>");
document.write("<TR><TH>Scope</TH>");
document.write("<TH>Value</TH></TR>");
document.write("<TR><TD>Global</TD>");
document.write("<TD>");
document.write(scope);
document.write("</TD></TR>");
document.write("<TR><TD>myFunc1</TD>");
document.write("<TD>");
document.write(myFunc1());
document.write("</TD></TR>");
```

```
document.write("<TR><TD>myFunc2</TD>");
document.write("<TD>");
document.write(myFunc2());
document.write("</TD></TR>");
document.write("</TABLE>");
```

See also:	Argument, Definition, Function call, Function call operator ( ), function(...) ..., Integer-value-remainder, Left-Hand-Side expression, Method, Parameter, Property

Cross-references:

ECMA 262 edition 2 section –11.2.3

ECMA 262 edition 2 section –11.2.4

ECMA 262 edition 2 section –13

ECMA 262 edition 3 section –11.2.3

ECMA 262 edition 3 section –11.2.4

ECMA 262 edition 3 section –13

Function arguments (Definition)

The parameters that are passed to a function as it is called.

When values are passed to a function as it is called, primitive values are passed by value. This means that from inside the function you see a copy of the value and cannot damage the one outside.

Objects are passed by reference. If you modify the properties of an object that is passed into a function by way of an argument, you will be modifying the master instance of that object because there is only one.

This means that calling a function can have side effects that you may not expect unless you realize that this is happening.

See also:	`Arguments` object, `arguments[]`, `Function.arguments[]`, Reference, Reference counting, Variable

Cross-references:

Wrox *Instant JavaScript* –page –27

Wrox *Instant JavaScript* –page –29

Function call (Definition)

A means of invoking a function definition.

A function may be invoked at any time. Typically in a web browser, functions are attached to event handlers belonging to document objects.

A function call consists of the identifier that names the function and a set of parentheses enclosing the optional parameters. The declaration of the function specifies the identifier and the formal parameter list.

The identifier must match letter for letter in the same case.

When a function is called either as a procedure or as part of an expression, there is the possibility of somewhat massive side effects to take place before the function returns. Indeed, it is possible to build a function that actually never returns.

Functions can be called in other documents, other frames and other windows. However, there are some security implications regarding whether those functions are accessible –the calling and called functions must exist in pages that were loaded from the same server or domain unless the security controls can be relaxed.

It is generally easier to call functions in parent windows and frames and then call downwards to their children. This allows some session state to be maintained in the parent's global scope. The parent can be a frameset and need not be a visible window or frame.

If you are using the parent as a means of maintaining state information, you might want to implement accessor functions to store and retrieve state information. This allows the validation of the values to be range checked in one place and for the physical storage implementation to be hidden. This is indicative of a good Object Oriented Programming style.

The result returned by a function call depends on the function script source text.

See also:	Arguments object, Calling event handlers, Function, Function code, Function object, Function property, Function prototype, function(...) ..., `Function()`, `Function()`, `Function.arguments[]`, `Function.arity`, `Function.Class`, `Function.constructor`, `Function.length`, `Function.prototype`, `Function.toString()`

Cross-references:

Wrox *Instant JavaScript* –page –27

Function call operator ( ) (Definition)

The parentheses change a property into a function/method.

- A property invoked as an `LValue` is used as an assignment target.
- A property invoked as an `RValue` returns its value, which for a function or method is the function declaration.
- A property invoked as an `RValue` with the parentheses and some optional arguments is called as a function or method.

The distinction between a function and method is whether the script source text in the function body makes use of the `'this'` keyword to operate on the receiving objects properties.

See also:	Arguments object, Function, Function code, Function literal, Function property, function(...) ..., `Function.arguments[]`, Grouping operator ( )

Function code (Definition)

Script source in a declared function.

Availability:	ECMAScript edition –2

Function code is the script source text that appears between the braces in a function declaration.

This could also fall under the heading of `eval` code if the function is being declared as a component within some script source text that has been passed to the `eval()` function for processing.

Function code initializes the scope chain to include its activation object followed by the `global` object.

Variable instantiation is performed using the activation object as the variable object and any initial variables are flagged with a `DontDelete` attribute.

The caller provides `this` value but in some situations the value `null` may be passed. In that case, the `global` object will be used in its place.

See also:	Compound statement, `eval()`, Executable code, Execution context, Function call, Function call operator ( ), function(...) ..., Script

Cross-references:

ECMA 262 edition 2 section –10.1.2

ECMA 262 edition 2 section –10.1.6

ECMA 262 edition 2 section –10.2.3

ECMA 262 edition 2 section –13

ECMA 262 edition 3 section –10.1.2

ECMA 262 edition 3 section –10.1.6

ECMA 262 edition 3 section –10.2.3

ECMA 262 edition 3 section –13

Function definition (Definition)

The description of a function in the script source text.

See also:	Declaration, Function call, Function call operator ( ), Parameter, `function( ... ) ...`

Function literal (Definition)

A way of creating functions on the fly within your script.

JavaScript version 1.2 introduces a genuine literal syntax for creating anonymous functions.

Previously we might have used a function constructor and assigned the result to an object. This is more simple and straightforward. It looks just like a normal function declaration, except that there is no name defined for the function.

Example code:

```
// An example function literal
var cube_value = function(a) { return a*a*a; };
```

See also:	Anonymous function, Function call operator ( ), function(...) ...

Function object (Object/core)

An object of the class "Function".

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server–2.0 Opera –3.0	
JavaScript syntax:	-	*myFunction* = Function
	-	*myFunction* = new Function()
Object properties:	arity, caller, constructor, length, prototype	
Object methods:	apply(), call(), toSource(), toString(), valueOf()	
Collections:	arguments[]	

An instance of the class "`Function`" is created by using the new operator on the `Function()` constructor. The new object adopts the behavior of the built-in prototype object through the prototype-inheritance mechanisms.

All properties and functions of the prototype are available as if they were part of the instance.

Many built-in objects are functions. They can be invoked with arguments. Some of these are constructors. They are functions that are intended to be used with the new operator.

Function objects come in four varieties according to how they are implemented. They may be built-in to the interpreter or may be provided as extensions in the script itself. The four types of functions are:

- Declared functions in script source text
- Anonymous functions build with the `Function` object constructor
- Implementation-supplied functions built into the host environment
- Internal functions built into the language

JavaScript has such relaxed syntax rules that it forgives the programmer if the arguments to a function are omitted. Instead, the interpreter will automatically pass the undefined value in place of the missing argument.

Every built-in function has the `Function` prototype object as the value returned by its internal `Prototype` property with the exception of the `Function` prototype object itself, which would return the `Object` prototype object.

The prototype for the `Function` prototype object is the `Object` prototype object.

If you want to create a function with no name, you can create your own `Function` object with the new operator. That function would have some script source associated with it and you can then call your function directly. Although it won't have a name, it would appear as if it did in the script source when you call it. However the name it would appear to have is actually the name of the variable containing the reference to it. Since it is an object, two variables can refer to the same object and you could call the same function under two different names. That might happen if you pass the function as an argument in the calling interface to another function. This is a way of implementing call-backs. You might build a comparator like this and pass it to a `sort()` function.

Creating function objects and referring to them in variables is somewhat like using an `eval()` function to execute script source that you have built in a string.

Example code:

```
// Create a function object
var myFunction = new Function("arg1, arg2", "return(arg1 * arg2);");

// And call it
document.write(myFunction(5, 10));
```

See also:

Aggregate type, Anonymous function, `Arguments.callee`, `Arguments.caller`, Built-in function, Declared function, Execution context, Function call, function(...) ..., Function(), `Function.arity`, `Function.Class`, `Function.length`, `Function.prototype`, Implementation-supplied function, Internal function, JavaScript to Java values, `Native` object, `Object` object, Parameter

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
arity	1.2 +	3.0 +	4.0 +	4.0 +	-	3.0 +	-	Warning, ReadOnly, DontDelete, DontEnum.
caller	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	2.0 +	-	Warning, ReadOnly, DontEnum., Deprecated
constructor	1.1 +	1.0 +	3.0 +	3.02 +	-	-	2 +	DontEnum.
length	1.1 +	1.0 +	4.0 +	3.02 +	-	-	2 +	Warning, ReadOnly, DontDelete, DontEnum., Deprecated
prototype	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	2.0 +	2 +	ReadOnly, DontDelete, DontEnum.

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
apply()	1.3 +	5.5 +	4.06 +	5.5 +	-	-	3 +	-
call()	1.3 +	5.5 +	4.06 +	5.5 +	-	-	3 +	-
toSource()	1.3 +	3.0 +	4.06 +	4.0 +	-	-	-	-
toString()	1.1 +	3.0 +	4.0 +	3.0 +	3.0 +	2.0 +	2 +	-
valueOf()	1.1 +	-	4.0 +	-	-	-	2 +	-

Cross-references:

ECMA 262 edition 2 section –10.1.1

ECMA 262 edition 2 section –10.2.4

ECMA 262 edition 2 section –13

ECMA 262 edition 2 section –15

ECMA 262 edition 2 section –15.3

ECMA 262 edition 3 section –10.1.1

ECMA 262 edition 3 section –13

ECMA 262 edition 3 section –15.3

O'Reilly *JavaScript Definitive Guide* –page –42

Function() (Constructor)

A Function object constructor.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0	
JavaScript syntax:	-	`new Function()`
	-	`new Function(`*`someArguments`*`)`
Argument list:	*`someArguments`*	Some formal parameters and a block of script source

The function constructor can be called with the new operator or as a function.

The initial value of `Function.prototype.constructor` is the built-in `Function` constructor.

The arguments supplied to the `Function()` constructor are all assumed to be parameters apart from the last one which is taken to be the body Source Script Text. If there is only one argument, then that is taken to be the body of the function.

If there are no arguments, an empty function is created.

Note that it is permissible but not necessary to have a separate argument for each formal parameter. All three of these examples produce exactly the same result:

- new Function("a", "b", "c", "return a+b+c")
- new Function("a, b, c", "return a+b+c")
- new Function("a, b", "c", "return a+b+c")

A prototype property is automatically created in case the function object is used as a constructor at some future time.

The function constructor will always create top level functions without static scoping. This is a little different to the behavior of a function literal, available in JavaScript version 1.2 onwards.

See also:	Constructor function, constructor property, Function call, Function object, Function.prototype, Global object, new, Object constant

Cross-references:

ECMA 262 edition 2 section –15.1.3.2

ECMA 262 edition 2 section –15.3.1

ECMA 262 edition 2 section –15.3.4.1

ECMA 262 edition 3 section –15.3.2

Function() (Function)

A Function object constructor.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	`Function()`
	-	`Function(`*`someArguments`*`)`
Argument list:	*`someArguments`*	Some formal parameters and a block of script source

When the `Function` constructor is called as a function, it creates and initializes a new function object. The function call `Function()` is equivalent to the expression new `Function()` with the same arguments.

The arguments supplied to the `Function()` constructor are all assumed to be parameters apart from the last one which is taken to be the body Source Script Text. If there is only one argument, then that is taken to be the body of the function.

See also:	Cast operator, Constructor function, constructor property, Function call, `Function.prototype`, Implicit conversion

Cross-references:

ECMA 262 edition 2 section –15.1.3.2

ECMA 262 edition 2 section –15.3.1

ECMA 262 edition 2 section –15.3.2.1

ECMA 262 edition 2 section –15.3.4.1

ECMA 262 edition 3 section –15.3.1

Function.apply() (Method)

Use a function object as if it belonged to another target object.

Availability:	ECMAScript edition –3 JavaScript –1.3 JScript –5.5 Internet Explorer –5.5 Netscape –4.06	
Property/method value type:	`Function result`	
JavaScript syntax:	-	`myFunction.apply(`*`anObject`*`, `*`anArray`*`)`
Argument list:	*`anArray`*	A set of arguments presented as an array
	`anObject`	An object to apply the function to

If you want to force a particular function implementation to be used on an object, you can locate the one you want and pass it an object on which to operate. The function then executes as if it were a property of the passed object instead of the receiving object.

This can be useful to restore some functionality that may have been overridden or to share a function definition between several objects.

This is effectively like adding the function as a property to the target object, executing it, passing the arguments to it and then deleting the function property afterwards. Applying a function is a lot easier.

This is similar to the `call()` method that passed the arguments as a comma separated list. This passes them as an array.

Example code:

```
// Use the fundamental valueOf method on an object that has overridden
// its own valueOf method
Object.prototype.valueOf.apply(myTargetObject);
```

See also:	`Element.applyElement()`

Cross-references:

ECMA 262 edition 3 section –15.3.4.3

Function.arguments[] (Collection)

The values passed to the function when it is called.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server–2.0 Opera –3.0 Deprecated	
JavaScript syntax:	-	*myFunction*.arguments

This property is only defined within a function body in a web browser. However some implementations may provide external arguments via this property.

Refer to the `Arguments` object topic for more details on how to access the caller and callee properties.

Warnings:

- This is an alternative way to access the arguments of a function. It is left over from earlier versions of JavaScript and although it still works, as of JavaScript version 1.2 it is deprecated. You should use the `arguments` object without there being any `function` object prefix to access it as a property.

See also:	Argument, Argument list, Arguments object, `Arguments.length`, `arguments[]`, Execution context, Function arguments, Function call, Function call operator ( ), function(...) ..., `Object.prototype`, Parameter

Property attributes:

`ReadOnly`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 section –10.1.6

ECMA 262 edition 2 section –10.1.8

ECMA 262 edition 2 section –15.2.3.1

ECMA 262 edition 3 section –10.1.6

ECMA 262 edition 3 section –10.1.8

Wrox *Instant JavaScript* –page –27

Function.arity (Property)

The number of arguments expected by a function call.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server–3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myFunction*.arity

This property yields a numeric value representing the maximum number of arguments the function expects to be called with and can support.

The `arity` property of the `Function` constructor returns 1 but it accepts a variable number of arguments.

Otherwise, the `arity` property returns a value that is typical for the function. Because the number of arguments can often be variable, this typical value should be used with caution.

This value is not necessarily the same as the length of the arguments array that is available inside the function when it is called. This indicates the actual number of arguments that were passed.

You can compare the two values to verify that the function was called with the correct arguments.

Warnings:

- There is a bug in the Netscape implementation of this property unless the `LANGUAGE` attribute is defined as "Javascript1.2" for the `<SCRIPT>` tag.

See also:	`Arguments.length`, Function call, Function object, `Function.length`, `Function.prototype`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 section –15.3.3.2

ECMA 262 edition 2 section –15.3.5.1

Function.call() (Method)

Uses a function object as if it belonged to another target object.

Availability:	ECMAScript edition –3 JavaScript –1.3 JScript –5.5 Internet Explorer –5.5 Netscape –4.06	
Property/method value type:	`Function result`	
JavaScript syntax:	-	*myFunction*`.call(`*anObject*`,` *anArgumentList*`)`
Argument list:	*anArgumentList*	A list of arguments
	anObject	An object to apply the function to

If you want to force a particular function implementation to be used on an object, you can locate the one you want and pass it an object on which to operate. The function then executes as if it were a property of the passed object instead of the receiving object.

This can be useful to restore some functionality that may have been overridden or to share a function definition between several objects.

This is effectively like adding the function as a property to the target object, executing it, passing the arguments to it and then deleting the function property afterwards. Applying a function is a lot easier.

This is similar to the `apply()` method which passed the arguments as an array. This passes them as a comma separated list.

Example code:

```
// Call this function against another object
myFunctionObject.call(myTargetObject, 100, "aaa");
```

See also:	Event handler scope

Cross-references:

ECMA 262 edition 3 section –15.3.4.4

Function.caller (Property)

A reference to the caller of the function.

Availability:	JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server–2.0 Opera –3.0 Deprecated
Property/method value type:	`Object object`
JavaScript syntax:	- *myFunction*`.caller`

Warnings:

- This is an alternative way to access the caller of a function. It is left over from earlier versions of JavaScript and although it still works, as of JavaScript version 1.2 it is deprecated. You should use the `arguments` object without there being any `function` object prefix to access it as a property and reference the caller from there.

See also:	`Arguments.caller`

Property attributes:

`ReadOnly`, `DontEnum`.

Function.Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition –2

This is an internal property that describes the class that a `Function` object instance is a member of. The reserved words suggest that in the future, this property may be externalised.

See also:	Class, Function call, `Function` object

Property attributes:

`DontEnum`, `Internal`.

Cross-references:

ECMA 262 edition 2 section –8.6.2

ECMA 262 edition 2 section –15.3.2.1

ECMA 262 edition 3 section –8.6.2

Function.constructor (Property)

A reference to a constructor object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myFunction*`.constructor`

The initial value of `Function.prototype.constructor` is the built-in `Function` constructor.

You can use this as one way of creating function objects although it is more popular to use the new `Function()` technique or to simply create them in the script source text.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	Function call, `Function.prototype`

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 section –15.3.4.1

ECMA 262 edition 3 section –15.3.2

Function.length (Property)

The number of arguments expected by a function call.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myFunction*`.length`

This property yields a numeric value representing the number of arguments the function expects to be called with and can support.

The `length` property of the `Function` constructor returns 1 but it accepts a variable number of arguments.

Otherwise the `length` property returns a value that is typical for the function. Because the number of arguments can often be variable, this typical value should be used with caution.

This value is not necessarily the same as the length of the arguments array that is available inside the function when it is called. This indicates the actual number of arguments that were passed.

Warnings:

- There is a bug in the Netscape implementation of this property unless the `LANGUAGE` attribute is defined as "JavaScript1.2" for the `<SCRIPT>` tag.
- In any case the `function.length` property is deprecated in favor of the `arity` property. This is to avoid confusion between functions and arrays.

See also:	`Arguments.length`, Function call, Function object, `Function.arity`, `Function.prototype`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 section –15.3.3.2

ECMA 262 edition 2 section –15.3.5.1

ECMA 262 edition 3 section –15.3.5.1

Function.prototype (Property)

The prototype for the `Function` object that can be used to extend the interface for all `Function` objects.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server–2.0 Opera –3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	`Function.prototype`
	-	*`myFunction`*`.constructor.prototype`

This is the prototype function object belonging to the `global` object and from which all `Function` objects are descended.

The initial value of `Function.prototype` is the built-in `Function` prototype object.

The value of the prototype property of a function is used to initialize child objects when that function is used as a constructor.

The following properties are inherited from the `Function.prototype`:

- `Function.prototype`
- `Function.constructor`

The following methods are inherited from the `Function.prototype`:

- `Function.toString()`

The following properties are provided by the instances of the `Function` object:

- `Function.length`

The example demonstrates how to provide extensions to all instances of this class by adding a function to the prototype object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that extends the output capabilities of Function objects
function sourceDump()
{
   myString = this.toString();
   myArray  = myString.split("{");
   myString = myArray.join("<BR>{<BR>");
   myArray  = myString.split("}");
   myString = myArray.join("<BR>}<BR>");
   return myString;
}

// Register the new function
Function.prototype.sourceDump = sourceDump;

// Create a function and test the Function.sourceDump() method
var myFunction = new Function("arg1, arg2", "return(arg1 * arg2);");
document.write(myFunction.sourceDump());
</SCRIPT>
</BODY>
</HTML>
```

See also:	Function call, Function object, `Function()`, `Function()`, `Function.arity`, `Function.constructor`, `Function.length`, `Function.toString()`, prototype property

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 section –15

ECMA 262 edition 2 section –15.3.2.1

ECMA 262 edition 2 section –15.3.3.1

ECMA 262 edition 2 section –15.3.4

ECMA 262 edition 2 section –15.3.5.2

ECMA 262 edition 3 section –15.3

Function.toSource() (Method)

Returns a string primitive version of the function.

Availability:	JavaScript –1.3 JScript –3.0 Internet Explorer –4.0 Netscape –4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFunction*`.toSource()`

This value is identical to the `toString()` method (exclusive to Netscape). In MSIE, it is not supported but since `toString()` is available in MSIE the recommended approach is to use `toString()` instead of `toSource()` for portability reasons.

Function.toString() (Method)

Returns a string primitive version of an object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –3.0 Internet Explorer –3.0 Netscape –4.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myFunction*`.toString()`

The result of calling this method is to obtain the source text of the function definition unless it is a host implemented function. In that case, an implementation dependent value will be returned by this method.

See also:	Cast operator, Function call, `Function.prototype`, `Function.valueOf()`,`toString()`

Cross-references:

ECMA 262 edition 2 section –15.3.4.2

ECMA 262 edition 3 section –15.3.4.2

Function.valueOf() (Method)

Returns a string primitive version of the function.

Availability:	ECMAScript edition –2 JavaScript –1.1 Netscape –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*`myFunction`*`.valueOf()`

This value is identical to the `toString()` method (exclusive to Netscape). The recommended approach is to use `toString()` instead of `valueOf()` for reasons of consistency.

The result of calling this method is to obtain the source text of the function definition unless it is a host implemented function.

See also:	`Function.toString()`

Cross-references:

ECMA 262 edition 2 section –15.3.4

Function object properties (Definition)

User defined properties added to a function object.

You can add your own properties to a function object. These can then be used as static variables inside the function. Since the function object persists, so do its properties. Variables declared inside the function with a `var` statement only persist while the function context is called and present in the scope chain.

See also:	Static method, Static variable

Function property (Definition)

A property belonging to an object which can be called as a function.

Availability:	ECMAScript edition –2

`Function` properties are implemented as `function` objects stored as the value of a named property within an object.

See also:	Call, `escape()`, `eval()`, Function call, Function call operator ( ), `isFinite()`, `isNaN()`, `parseInt()`, `unescape()`

Cross-references:

ECMA 262 edition 2 section –15.1.2

ECMA 262 edition 3 section –15.1.2

Function prototype (Definition)

A description of the calling interface to a function.

Warnings:

- In C language, the declaration of a function may also include a function prototype. This defines the calling interface to the function so that the compiler can give warning about any function calls that do not conform to the prototype.
- JavaScript does not support this function prototyping because it is not a strongly typed language and the scripts are not compiled before execution. Do not confuse function prototype calling interface specifications with the JavaScript `Function.prototype` object, which is part of the inheritance mechanism.

See also:	const, Function call, function(...) ...

Function scope (Definition)

The context within which the body script runs is the function scope.

JavaScript uses static scoping. This means that the functions are executed in the scope in which they are defined. This is important because any global values they refer to belong to the page in which they live. That is not necessarily the page that they were called from. This is only likely to cause any confusion when a multiple framed document is being rendered.

The function scope is the state of the scope chain as it was when the function was called, plus the `call` object added onto the end of the scope chain. The `call` object is the function being executed.

Because the `call` object is added to the scope chain, you don't need to refer to it explicitly and the current context inherits all its properties and methods.

Warnings:

- The static properties of a regular expression object do not conform to the same static scoping rules. Their static or class-based properties are dynamically scoped and available in the scope chain from which they are executed.

See also:	Event handler scope, `RegExp` object

function(...) ... (Declaration)

The description of a function in the script source text.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server–2.0 Opera –3.0	
JavaScript syntax:	-	`function` *`anIdentifier`* `(`*`aParameter`*`, ...) { scriptSource }`
Argument list:	*`anIdentifier`*	The name of the function being declared
	`aParameter`	One of the formal parameters to be passed to the function when it is called
	`scriptSource`	The source text for the script code that is executed when the function is called

A function declaration is a description of a function in the script source text. It provides the function name and a list of its arguments. It also provides a block of script code to be executed when the function is called.

Functions can be declared in the script source to add to the functions your script can make use of. When they are called, the prototype inheritance mechanism matches most local instances of the function having the name that the caller requests. This provides a way to override methods in parent prototypes or the `global` object.

When functions are declared in `global` and `eval` code, the new function object that is instantiated is added to the `variable` object for the owner of the script source text. It uses the function identifier as a name for the dictionary entry in the `variable` object. The functions are added in the order in which they appear in the Script Source Text and will replace any previously existing entry. Attributes are set according to the type of code being evaluated.

New functions can be added to the scripting environment as needed. They are described in the source script text with function declarators.

A function is declared with the function keyword, a set of parentheses enclosing its passed arguments and a block of executable code enclosed in curly braces.

Functions will always return a result but a function call can be cast to a void type to discard the resulting value. If you don't indicate a result to return yourself, the function mechanism would return the value undefined.

The internal mechanics of function declaration is to add a function property to the `global` object whose name is the function's identifier. The value of that property is a function object with the given parameter list and statement block.

If the function definition is part of the source text supplied to an `eval` function, then the function may be added to an internal activation object rather than the `global` object.

The act of executing or invoking a function is to call it.

Calling a function is accomplished by using the function property of an object as an `RValue` and appending the parentheses grouping operators with option arguments grouped within them.

Functions can be created and their script can refer to objects that don't yet exist. However, you may not run them until the objects they refer to have been created. This is often the case with event handlers that are defined speculatively on the basis that they may be needed but in fact they may never be called at all.

As of JavaScript version 1.2, you can declare new functions anywhere in a script source text. Prior to this, you could only define them in `global` code but not inside any `if()` blocks, `loop` blocks or `with` blocks. In JavaScript version 1.2, functions can be declared inside these contexts and inside functions themselves. This means that function availability can be localized to only be usable within a function's context scope.

Warnings:

- Debugging namespace collisions between function and variable names can be difficult to debug. For very large projects with many people working on shared code, you may want to establish some strict naming conventions to partition the namespace.
- This is an issue because function declarations occur before variable declarations. If you declare a variable with the same name as a function and assign a value to it the function will be inaccessible since you will have replaced the reference to its object with the value you just assigned to the variable.
- This will still happen even if the `var` declaration is placed sequentially earlier in the script block than the function declaration. The first pass declares and sets up the function. The second pass instantiates the variable.
- Remember that functions are created at compile (parse) time, and variables at run-time.

Example code:

```
// Here is an example function declaration:
function circularArea (aRadius)
{
   someGlobalValue = aRadius;
   return(Math.pow(aRadius,2)*Math.PI);
}
```

```
// We are using functions belonging to the Math object to
// raise the passed in radius value to the power of 2
// and multiply the result by the constant value of PI.
// We can call this and assign its value like this:
myArea = circularArea(12);
alert(myArea);

// If we simply wanted to execute the function and discard its
// result we might use this form:
void circularArea(10000);

// The conseqence is just to set the global value but we don't
// do anything with the returned value.
```

See also:	Formal Parameter List, `Function.arguments[]`

Cross-references:

ECMA 262 edition 2 section –10.1.1

ECMA 262 edition 2 section –10.1.3

ECMA 262 edition 2 section –10.1.6

ECMA 262 edition 2 section –13

ECMA 262 edition 2 section –15.3.2.1

ECMA 262 edition 3 section –10.1.1

ECMA 262 edition 3 section –13

ECMA 262 edition 3 section –15.3.2.1

Wrox *Instant JavaScript* –page –26

Fundamental data type (Definition)

The basic simple native data types.

The fundamental data types in JavaScript are:

- Number
- String
- Boolean

All other data types are expressed in terms of these kinds of value.

See also:	Boolean, Number, Regular expression, String

Furniture (Definition)

A name for the various items that form the window border.

Refer to:

`Window furniture`

Garbage collection (Definition)

The mechanism by which deallocated strings and objects are cleared from memory.

Garbage collection happens in JavaScript, but you have very little control over when and how. Since you don't have pointers, you cannot identify items that need to be released and nor do you need to.

Garbage collection happens in a web browser implementation when the page is reloaded. Any objects, strings, functions and variables you were using are discarded and no longer available. If you want to keep them around, you need to put them somewhere else, possibly in a parent `FRAMESET` object or as member properties of the `Navigator` object although that's not very portable.

The old JavaScript context is completely destroyed and a new one is made from scratch. This means that each page owns a pool of memory from which its scripts allocate storage. The pool is flushed on a page by page basis. This may get a little more complicated when you have scripts running across frames, since some pools will get flushed and others won't –be careful that the top-most frame does not become a memory hog, as this is the only one you cannot purge without losing all the session state. You could store the state data in cookies or in a transitory page in a separate window and then fetch it back again after the frame-set is regenerated. However, this "solution" is a bit of a hack, and it's better to solve the problem at source rather than try to hide the consequences.

There are many garbage collection algorithms, some better than others, but since you have no control over garbage collection, there is little point in investigating it, other than to influence the strategy you might use for creating and destroying objects. Some techniques may yield better performance in certain browsers due to the garbage collection algorithm the browser uses. Since the browser implementers tend to change this from time to time, there is no guarantee that a technique you deploy now will be optimal for the next version of the same browser.

However, you should try to avoid creating and destroying large numbers of strings and objects, since this seems to lead to memory leakage on most platforms. MSIE version 4 on Windows is particularly prone to this and it's not hard to develop animation loops that consume 50K of memory on each cycle. Virtually any elimination of String construction/destruction is worthwhile in cyclic situations. Indeed, you might find that `innerText` properties are a better place to store textual content than strings if things become extreme.

See also:	delete, Execution environment, Memory leak, Memory management, `Object()`, `Option()`, Reference counting, Variable

Cross-references:

Wrox *Instant JavaScript* –page –29

Get() (Function/internal)

Internal private function that is used to access public properties.

Availability:	ECMAScript edition –2

This internal function is used to retrieve internal properties from objects.

If the property exists in the receiving object, its value will be returned.

If the property is not a member of the receiving object, then if the `Prototype` property for this object returns null, we have reached the top of the prototype chain so the property is undefined. The result will be the undefined value.

If the property does not exist, and there is a parent `Prototype` object, then the message is passed to that object for evaluation.

The `Get` internal function may indeed return a value when received by a host object, even if that host object would respond to the `HasProperty` function with a false result indicating that the property does not exist.

See also:	Accessor method, `GetValue()`, Internal Method

Cross-references:

ECMA 262 edition 2 –section –8.6.2.1

ECMA 262 edition 3 –section –8.6.2.1

GetBase() (Function/internal)

Internal private function.

Availability:	ECMAScript edition –2

This internal function returns the base object component pointed at by the reference item passed as its argument.

A runtime error will be generated if the passed in object is not a reference item.

See also:	Reference

Cross-references:

ECMA 262 edition 2 –section –8.7.1

ECMA 262 edition 3 –section –8.7.1

getClass() (Function)

Returns the `JavaClass` of a `JavaObject`.

Availability:	JavaScript –1.1 Netscape version –3.0	
JavaScript syntax:	N	*myJavaObject*.getClass()

Refer to:

`JavaObject.getClass()`

GetObject() (Function)

A JScript function that returns a reference to an object representing a file belonging to an application on your system.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	Automation object	
JavaScript syntax:	IE	GetObject(*aLocation*)
	IE	GetObject(*aLocation*, *anObjectType*)
	IE	GetObject(*aLocation*!*aSubObject*)
	IE	GetObject(*aLocation*!*aSubObject*, *anObjectType*)
Argument list:	*anObjectType*	What sort of application and object class type to be created
	aLocation	A path to the file for the object to be instantiated
	aSubObject	A fragment identifier for a sub-object within the file

When this function is called, the application may be activated to provide a remote interface to the file. You can also specify fragments within the file.

The path argument points at the file within the file system where the object you want reposes.

This is related to the `ActiveXObject()` constructor which creates an object that points at an application or document without loading a file to instantiate it.

The objects created by this function are called `Automation` objects.

The location value can have a fragment identifier delimited by an exclamation mark. With this, for example, you can refer to one worksheet within an Excel document.

Example code:

```
// Locate an Excel spreadsheet
myWorkbook = GetObject("F:\\DOCUMENTS\\ACCOUNTS.XLS");
// Locate one worksheet within an Excel spreadsheet
myWorkSheet = GetObject("F:\\DOCUMENTS\\ACCOUNTS.XLS!sheet4");
```

See also:	ActiveXObject()

GetPropertyName() (Function/internal)

Internal private function.

Availability:	ECMAScript edition –2

This internal function returns the property name component of the reference item passed in its argument.

A run time error is generated if the argument passed is not a reference item.

See also:	Reference

Cross-references:

ECMA 262 edition 2 –section –8.7.2

ECMA 262 edition 3 –section –8.7.2

GetValue() (Function/internal)

Internal private function.

Availability:	ECMAScript edition –2

This internal function returns the value contained in the property belonging to the object pointed at by the reference item passed in its argument.

If the passed-in argument is not a reference item, it is returned as the result.

If the object being referred to does not exist, a run-time error is generated.

Otherwise the usual `Get()` function behavior is invoked for the property named in the reference.

See also:	`Get()`, Reference

Cross-references:

ECMA 262 edition 2 –section –8.7.3

ECMA 262 edition 3 –section –8.7.1

Global code (Definition)

Script source that is outside of any function code blocks.

Availability:	ECMAScript edition –2

Global code is that source text that is outside of all function declarations.

On initialization of some global code into an execution context, the scope chain is initialized to only contain the global object and nothing else.

Variable instantiation is performed using the global object as the variable object and with empty property attributes.

This value is set to point at the global object.

See also:	Executable code, Execution context, `function(…)… `, Script

Cross-references:

ECMA 262 edition 2 –section –10.1.2

ECMA 262 edition 2 –section –10.2.1

ECMA 262 edition 3 –section –10.1.2

ECMA 262 edition 3 –section –10.2.1

Global object (Object/core)

A special type of object that is always available in the prototype chain.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.0 Netscape –2.0	
JavaScript syntax:	-	*myGlobal* = document.parentWindow
	-	*myGlobal* = frame
	-	*myGlobal* = self
	-	*myGlobal* = window
	-	*myGlobal* = window.frames[*anIndex*]
Argument list:	*anIndex*	A reference to a window object which is also a global object for that window
Object properties:	clientInformation, clipboardData, closed, crypto, defaultStatus, dialogArguments, dialogHeight, dialogLeft, dialogTop, dialogWidth, document, event, external, frame, frameRate, history, innerHeight, innerWidth, java, length, location, locationbar, Math, menubar, name, navigator, netscape, offScreenBuffering, opener, outerHeight, outerWidth, Packages, pageXOffset, pageYOffset, parent, personalbar, pkcs11, returnValue, screen, screenLeft, screenTop, screenX, screenY, scrollbars, secure, self, status, statusbar, sun, toolbar, top, window	
Class constants:	Infinity, NaN, undefined	
Object methods:	addClient(), addResponseHeader(), alert(), Array(), attachEvent(), back(), blob(), blur(), Boolean(), callC(), clearInterval(), clearTimeout(), close(), confirm(), Date(), debug(), deleteResponseHeader(), detachEvent(), disableExternalCapture(), enableExternalCapture(), escape(), eval(), execScript(), find(), flush, focus(), forward(), Function(), getOptionValue(), getOptionValueCount(), home(), isFinite(), isNaN(), moveBy(), moveTo(), navigate(), Number(), Object(), open(), parseFloat(), parseInt(), print(), prompt(), redirect, registerCFunction(), resizeBy(), resizeTo(), scroll(), scrollBy(), scrollTo(), setHotkeys(), setInterval(), setResizable(), setTimeout(), setZOptions(), showHelp(), showModalDialog(), showModelessDialog(), ssjs_generateClientID(), ssjs_getCGIVariable(), ssjs_getClientID(), stop(), String(), typeof(), unescape(), write()	
Functions:	atob(), btoa(), captureEvents(), handleEvent(), releaseEvents(), routeEvent()	
Event handlers:	onAfterPrint, onBeforePrint, onBeforeUnload, onBlur, onDragDrop, onError, onFocus, onHelp, onLoad, onMove, onResize, onUnload	
Collections:	frames[]	

The `Global` object is unique and is created before any code is executed. In an ECMAScript compliant implementation it is where variables, methods, functions and properties are stored if you don't explicitly attach them to another object yourself. The member properties, methods, functions and variables are globally available because the `Global` object is placed into the scope chain of every execution context.

In a web browser, the `Global` object is also the `Window` object. In a server-side implementation, the `Global` object is probably the `Response` object but this is not mandatory. Other implementations will use one of the host objects as the `Global` object but there is no defined and standard choice.

Since you don't ever use the keyword `Global` and can avoid using the keyword `Window`, the properties and methods look as if they are part of the core language rather than members of an object. Web browsers are becoming more ECMAScript compliant as new versions are released.

The `Global` object is added to the scope chain of a program when it commences execution. Other built-in objects are accessible as initial properties of the `Global` object. Some of these are added as core functionality and are available in all implementations. Others are added as host objects defined differently for each implementation.

When the `Global` object is created, it always has at least the following properties:

- `Object` object
- `Function` object
- `Array` object
- `String` object
- `Boolean` object
- `Number` object
- `Date` object
- `Math` object
- Value properties
- Utility function properties
- Additional host defined properties

Most of the properties initially provided by the `Global` object cannot be enumerated as their `DontEnum` attribute is set (at least in ECMA-compliant implementations).

The initial properties soon change and are added to when the flow of control enters execution contexts and begins to process scripts.

The value properties that are initially set up in the `Global` object are listed in this table:

Name
`Infinity`
`NaN`

Refer to the discussion on each of value properties for more details.

Some host implementations create aliases for the `Global` object and store those self-referring aliases in the `Global` object when it is initialized. An example of this behavior is the `window` object created in some web browser host implementations of JavaScript.

There is one `Global` object for each window in a web browser. There is also one for each frame in a frame-set. This means that you can have `Global` objects that have global variables that are not shared with one another.

However, you can access them by referring to the parent object. This goes to an object context that contains the current frame. You can also access the top or outer window of a frame-set hierarchy directly.

Whether you can truly access objects in other windows or frames comes down to some basic security issues. Generally the security policy prevents access to code that arrives from different servers. You can legally get at values in pages served from the same host as your page was served from. It is unreasonable to be able to access variables in a page served from another host. This can lead to difficulties if you serve your site from multiple hosts. It isn't as limiting as all that though and there are ways to grant permission via the security policy in the browser.

`Global` objects support different sub-sets of the complete range of properties according to the context in which they are used. Server-side JavaScript won't support anything to do with window display and client-side (browser) JavaScript does not need to know about request-response handling.

If you want to access a `Global` object in another context, perhaps in a different window, you need to assign it to a variable. In the syntax listing this is illustrated by assigning references to `Global` objects to the *`myGlobal`* variable.

Warnings:

- You cannot use the `Global` object with the new operator to make a copy.
- You cannot call the `Global` object as a function.
- The value of the internal Prototype property of the `Global` object is implementation dependent. Since it contains the `Object` object as one of its children, it could not inherit its Prototype from there since that could set up an endlessly recurring loop of Prototype inheritors. It is likely that the Prototype of the `Global` object is `Null`. If it is important that you know the value of this, then you should test it with a debugging script first. However, you may be writing non-portable code if you depend on the value that it indicates.
- In a web browser, the Window object serves as the `Global` object. It owns all the properties that you would expect the `Global` object to have in a core implementation. The hosting environment adds the other window-based properties.

See also:	`escape()`, `eval()`, Infinity, `isFinite()`, `isNaN()`, `NaN`, `parseFloat()`, `parseInt()`, typeof, undefined, `unescape()`, `Window` object

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`clientInformation`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`clipboardData`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`closed`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`crypto`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`defaultStatus`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
dialogArguments	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
dialogHeight	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
dialogLeft	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
dialogTop	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
dialogWidth	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
document	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
event	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
external	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
frame	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
frameRate	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
history	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
innerHeight	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
innerWidth	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
java	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
length	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
location	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
locationbar	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Math	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
menubar	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
name	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
navigator	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
netscape	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
offScreenBuffering	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
opener	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
outerHeight	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
outerWidth	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Packages	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
pageXOffset	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
pageYOffset	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
parent	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
personalbar	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
pkcs11	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
returnValue	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
screen	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
screenLeft	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
screenTop	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
screenX	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
screenY	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
scrollbars	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
secure	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
self	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
status	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
statusbar	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
sun	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
toolbar	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
top	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
window	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
addClient()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
addResponse Header()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
alert()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Array()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
attachEvent()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
back()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
blob()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
blur()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Boolean()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
callC()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
clearInterval()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
clearTimeout()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
close()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
confirm()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Date()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
debug()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
delete ResponseHeader()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
detachEvent()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
disableExternal Capture()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
enableExternal Capture()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
escape()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
eval()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
execScript()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
find()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
flush	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
focus()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
forward()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Function()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
getOptionValue()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
getOption ValueCount()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
home()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
isFinite()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
isNaN()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
moveBy()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
moveTo()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
navigate()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Number()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
Object()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
open()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
parseFloat()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
parseInt()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
print()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
prompt()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
redirect	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
register CFunction()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
resizeBy()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
resizeTo()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
scroll()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
scrollBy()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
scrollTo()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
setHotkeys()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
setInterval()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
setResizable()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
setTimeout()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
setZOptions()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
showHelp()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
showModalDialog()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
showModeless Dialog()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
ssjs_generate ClientID()	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`ssjs_get CGIVariable()`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`ssjs_getClientID()`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`stop()`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`String()`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`typeof()`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`unescape()`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning
`write()`	1.0 +	1.0 +	2.0 +	3.0 +	-	-	2 +	Warning

Event name	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`onAfterPrint`	-	5.0 +	-	5.0 +	-	-	-	-
`onBeforePrint`	-	5.0 +	-	5.0 +	-	-	-	-
`onBeforeUnload`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onDragDrop`	1.2 +	-	4.0 +	-	-	-	-	-
`onError`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onLoad`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
`onMove`	1.2 +	-	4.0 +	-	-	-	-	-
`onResize`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
`onUnload`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning

Cross-references:

ECMA 262 edition 2 –section –10.1.5

ECMA 262 edition 2 –section –15

ECMA 262 edition 3 –section –10.1.5

ECMA 262 edition 3 –section –15.1

Global.undefined (Constant/static)

A globally available copy of the undefined value.

Availability:	ECMAScript edition –2 JavaScript –1.3 JScript –5.5 Internet Explorer –5.5 Netscape –4.06
Property/method value type:	`Undefined primitive`

See also:	`Global` object, undefined

Cross-references:

ECMA 262 edition 2 –section –4.3.9

ECMA 262 edition 3 –section –4.3.9

Global special variable (Definition)

Special variables that are globally available.

Availability:	ECMAScript edition –2

Some special variables are created in the interpreter to assist with program execution and to define certain special values. You must avoid using these as identifiers.

Here is a list of the special variable names:

Name
`Infinity`
`NaN`

Some implementations may allow you to redefine the value of these special variables. This can cause unpredictable side effects later on.

See also:	`Global` object, `Infinity`, `NaN`

Cross-references:

ECMA 262 edition 2 –section –15.1.1

ECMA 262 edition 3 –section –15.1.1

Glow() (Filter/visual)

A visual filter for adding a glow effect

Availability:	JScript –3.0 Internet Explorer –4.0

Refer to:

Filter – `Glow()`

Glue code (Definition)

Supporting code to integrate JavaScript with the environment.

This so called glue code is that which provides a conduit for signals to be sent between different parts of the hosting environment. Most likely as far as JavaScript is concerned, this would be a mechanism for transmitting events to the interpreter so that a particular handler function can be activated.

There are many proprietary names for these mechanisms, but fundamentally they all do the same job, which is to connect the script to the environment. Here are some examples of enabling technologies that may be involved to a greater or lesser extent in some platforms:

- ActiveX
- LiveConnect
- AppleEvents
- Signals
- BSD Sockets
- Call-backs from plugins

See also:	ActiveX, ECMAScript, Host environment, LiveConnect, Plugin object

Cross-references:

Wrox *Instant JavaScript* –page –12

Gotcha (Definition)

A pitfall or catch-out for the unwary and the experienced script developer.

Refer to:

Pitfalls

goto (Reserved word)

Reserved for future language enhancements.

This keyword suggests that future versions of JavaScript may support the `goto` statement which will unconditionally go to a labelled portion of script source text. This also suggests that labels will need to be supported as well.

ECMA edition 3 already mandates that case and default labels are supported for the benefit of the `switch()` statement.

The ECMA standard notes that although it is reserved future use, an implementation is still compliant if it provides the appropriate functionality of these reserved keywords.

See also:	Jump statement, Label, Reserved word, Reserved Word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Gradient() (Filter/visual)

A procedural definition of a gradient effect.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `Gradient()`

GradientWipe() (Filter/transition)

A transition effect with the appearance of a wipe with a soft edge.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `GradientWipe()`

Grayscale() (Filter/visual)

A visual filter for converting to a grayscale appearance.

Availability:	JScript –3.0 Internet Explorer –4.0

Refer to:

Filter – `Grayscale()`

Greater than (>) (Operator/relational)

Compares two operands to determine which is nearer to +Infinity.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`anOperand1 > anOperand2`*
Argument list:	*`anOperand1`*	A value that can be compared numerically or lexically
	`anOperand2`	A compatible value

Returns `true` if the left operand is numerically greater than the right operand or is sorted later in the Unicode collating sequence when two string values are compared.

In numeric comparisons, the presence of `NaN` in either or both operands will yield undefined instead of `true` or `false`.

When comparing two strings, a prefixing plus sign is present, then a numeric coercion of a string takes place before the comparison. Numeric coercion takes place when either of the operands is numeric.

In ECMA compliant JavaScript implementations, string values are simply compared according to the Unicode character code point values, with no attempt to provide the more complex semantically oriented definitions of character and string equality defined in the Unicode version 2.0 specification.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

The result is the Boolean value `true` if *`anOperand1`* is numerically or lexically greater than *`anOperand2`*; otherwise `false` is returned.

See also:	ASCII, Associativity, Equal to (==), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), Logical expression, Logical operator, NOT Equal to (!=), NOT Identically equal to (!==), Operator Precedence, Relational expression, Relational operator, Unicode

Cross-references:

ECMA 262 edition 2 –section –11.8.2

ECMA 262 edition 2 –section –11.8.5

ECMA 262 edition 3 –section –11.8.2

ECMA 262 edition 3 –section –11.8.5

Greater than or equal to (>=) (Operator/relational)

Compare two operands to determine which is nearer to +Infinity or whether they are equal.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`anOperand1 >= anOperand2`*
Argument list:	*`anOperand1`*	A value that can be compared numerically or lexically
	`anOperand2`	A compatible value

Returns true if the left operand is numerically greater than or equal to the right operand or is sorted later or identically in the Unicode collating sequence when two string values are compared.

In numeric comparisons, the presence of `NaN` in either or both operands will yield undefined instead of `true` or `false`.

When comparing two strings, a prefixing plus sign is present, then a numeric coercion of a string takes place before the comparison. Numeric coercion takes place when either of the operands is numeric.

In ECMA compliant JavaScript implementations, string values are simply compared according to the Unicode character code point values, with no attempt to provide the more complex semantically oriented definitions of character and string equality defined in the Unicode version 2.0 specification.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

The result is the Boolean value `true` if *`anOperand1`* is numerically or lexically greater than or equal to *`anOperand2`*; otherwise `false` is returned.

See also:	ASCII, Associativity, Equal to (==), Greater than (>), Identically equal to (===), Less than (<), Less than or equal to (<=), Logical expression, Logical operator, NOT Equal to (!=), NOT Identically equal to (!==), Operator Precedence, Relational expression, Relational operator, Unicode

Cross-references:

ECMA 262 edition 2 –section –11.8.4

ECMA 262 edition 3 –section –11.8.4

Grouping operator ( ) (Delimiter)

A means of controlling precedence of evaluation in expressions.

Availability:	ECMAScript edition –2

The grouping operator is a pair of parentheses placed around an expression or expressions to control the precedence of evaluation in expressions so that the sub-expressions are evaluated in the correct order. It is also used to enclose the arguments to a function or method.

Placing parentheses around expressions controls the order in which they are evaluated and can override the normal precedence that operators assume. This allows `delete` and `typeof` operations to be applied to expressions in parentheses for example.

Controlling the precedence of expressions allows operators with lower precedence to be evaluated ahead of the higher priority expression operators. For example:

```
A + B * C
```

by implication is executed like this:

```
A + (B * C)
```

A and B can be added before the multiplication like this:

```
(A + B) * C
```

This forces the addition to occur before the multiplication and is functionally equivalent to:

```
A*C + B*C
```

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

See also:	Associativity, delete, Function call operator ( ), Operator, Operator Precedence, Parentheses ( ), Primary expression, typeof

Cross-references:

ECMA 262 edition 2 –section –11.1.4

ECMA 262 edition 3 –section –11.1.6

handleEvent() (Function)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`handleEvent(`*`anEvent`*`)`
	N	*`myWindow`*`.handleEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An `Event` object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

This method is supported by all objects that respond to events. It is part of the event management suite which allow events to be routed to handlers other than just the one that defaults to being associated with an event.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`Button.handleEvent()`, `captureEvents()`, `Checkbox.handleEvent()`, `Document.handleEvent()`, Event handler, Event management, `FileUpload.handleEvent()`, `Form.handleEvent()`, `Input.handleEvent()`, `Layer.handleEvent()`, `Password.handleEvent()`, `RadioButton.handleEvent()`, `ResetButton.handleEvent()`, `Window.handleEvent()`, `Window.routeEvent()`

Cross-references:

Wrox *Instant JavaScript* - page 55

Handler (Definition)

An event handler can be attached to an HTML tag as an attribute.

The event handlers can be attached to a particular object within the HTML document or within JavaScript. Some event handlers can only be attached with HTML tag attributes. Others only by setting the event handler property of an object to point at an event handling function.

The names of the handlers in HTML are case-insensitive, although there are some conventions.

The value associated with the event handling tag attribute is a fragment of JavaScript code. The script source text forms the body of an anonymous function object. This may simply contain a call to the name of an event handling function or it may be several lines of JavaScript code. It is probably a good idea to encapsulate the handler into a function and refer to it by name.

HasInstance() (Function/internal)

Internal private function to test for the existence of an instance.

Availability:	ECMAScript edition - 3

This internal function returns a Boolean value indicating whether the `function` object is the prototype of an instance object that is passed as an argument. If it is, then the function returns `true`.

This is an internal function very similar to the `Object.isPrototypeOf()` method.

See also:	`Object.isPrototypeOf()`

Cross-references:

ECMA 262 edition 3 section - 15.3.5.3

HasProperty() (Function/internal)

Internal private function to test for the existence of a property.

Availability:	ECMAScript edition - 2

This internal function returns a Boolean value indicating whether the object contains the named property.

If the receiving object has the property, then the result is `true`.

If the receiving object does not, then the prototype chain is walked until the property is found or the chain is exhausted.

If a `null` prototype is found, the `false` value is returned.

Host objects may or may not strictly honor the intent of this internal function. The ECMA standard allows for the possibility that a host object may still properly manage `Get` and `Put` internal functions, even if the `HasProperty` function returns `false` for the properties being accessed.

To cope with that eventuality, ECMAScript edition 3 provides a way to test whether an object has a property of its own, but this test ignores the prototype inheritance. You might simulate that by walking up the prototype chain, testing objects as you go.

See also:	`Attribute` object, `Attributes` object, Host object, Internal Method, `Object.hasOwnProperty()`

Cross-references:

ECMA 262 edition 2 section - 8.6.2.4

ECMA 262 edition 3 section - 8.6.2.4

HEAD object (Object/HTML)

A special MSIE object that represents the head block of an HTML document.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myHEAD* = *myDocument*.all.*anElementID*
	IE	*myHEAD* = *myDocument*.all.tags("HEAD")[*anIndex*]
	IE	*myHEAD* = *myDocument*.all[*aName*]
	-	*myHEAD* = *myDocument*.getElementById(*anElementID*)
	-	*myHEAD* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myHEAD* = *myDocument*.getElementsByTagName("HEAD")[*anIndex*]
HTML syntax:	`<HEAD> ... </HEAD>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`profile`, `vAlign`	
Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`	

This object inherits from HTML. It has a `TITLE` object as one of its children and its sibling is the `BODY` object that represents the contents of the `<BODY>` tag.

The `CLASS` tag attribute is supported by MSIE but serves no purpose other than maintaining consistency. You should avoid its use even though the reference is maintained within the corresponding `HEAD` object.

See also:	`BODY` object, `Element` object, `HEAD.profile`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`profile`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`vAlign`	-	3.0 +	-	4.0 +	-	-	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

HEAD.profile (Property)

The `profile` property of the current document's `<HEAD>` block.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 5.0 Internet Explorer - 5.0 Netscape - 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myHEAD*`.profile`
HTML syntax:	`<HEAD PROFILE="...">`	

This value should contain the value of the PROFILE attribute of the document's <HEAD> block. The PROFILE value is a means of sharing meta-data profiles between many documents. It's very like an include file for meta information, but does not get used very much. This property should contain a URL value for a shared meta-data profile.

It might be useful to be able to access the internals of this file, but the property only yields the URL that reaches the file on the server or local file system.

See also:	HEAD object

HEAD.vAlign (Property)

The vertical alignment associated with the <HEAD>.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myHEAD*.vAlign

Although this property is present and accessible, it should not affect the display of the page at all.

Warnings:

- This property is probably an implementation mistake. Although it is visible when the properties of the HEAD object are enumerated, it has no use in that context.

Hexadecimal value (Definition)

A numeric value based on a radix of 16.

A hexadecimal value is an integer composed of only the following characters:

```
0 1 2 3 4 5 6 7 8 9

A B C D E F

a b c d e f
```

Note the use of the alphabetic characters to extend the decimal number digit set.

Hexadecimal values are always prefixed by a zero and X character.

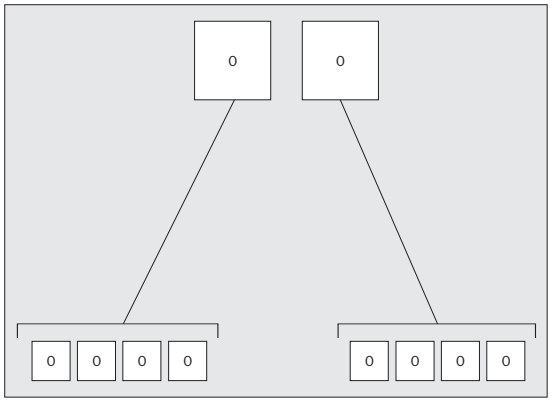
The sequence carries over for the next increment when each column reaches the value F. Thus:

```
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07

0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F

0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17
```

Hexadecimal values have a historical significance, having been used in the earliest computer systems. However, these days they are particularly useful since they map quite conveniently to the binary system. Each hexadecimal digit corresponds to four binary digits. Two hex digits map to a byte and four to a word. This is particularly useful and is evidenced by hexadecimal values being used to generate Unicode escape sequences.

There are several hexadecimal values that are useful. These are summarized in the following table:

Value	Description
0x0	All bits clear
0x20	The single bit to toggle between upper and lower case letters
0x7F	A seven bit character mask
0x8000	The sign bit of a 16 bit integer value
0x80000000	The sign bit of a 32 bit integer value
0xDF	A mask that excludes the upper/lower case bit
0xFF	All bits set in a byte
0xFFFF	All bits set in a 16 bit word
0xFFFFFFFF	All bits set in a 32 bit word



See also:	Decimal value, Integer constant, Number, `Number.toString()`, Octal value

Cross-references:

O'Reilly *JavaScript Definitive Guide* - page 35

Hidden object (Object/DOM)

A field of data submitted with the form but not visible to the user.

Availability:	DOM level - 1 JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Inherits from:	`Input object`	
JavaScript syntax:	-	*myHidden* = *myDocument*.*aFormName*.*anElementName*
	-	*myHidden* = *myDocument*.*aFormName*.elements[*anItemIndex*]
	IE	*myHidden* = *myDocument*.all.*anElementID*
	IE	*myHidden* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*myHidden* = *myDocument*.all[*aName*]
	-	*myHidden* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*myHidden* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*]
	-	*myHidden* = *myDocument*.getElementById(*anElementID*)
	-	*myHidden* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myHidden* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
HTML syntax:	`<INPUT TYPE="hidden">`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The `NAME` attribute of an element
	anElementID	The `ID` attribute of an element
	anItemIndex	A valid reference to an item in the collection
	aFormIndex	A reference to a particular form in the `forms` collection
Object properties:	`type`, `value`	
Event handlers:	`onAfterUpdate`, `onBeforeUpdate`, `onHelp`, `onRowEnter`, `onRowExit`	

Many properties, methods and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

There isn't really a `Hidden` object class but it is helpful when trying to understand the wide variety of input element types if we can reduce the complexity by discussing only the properties and methods of a hidden field. In actual fact, the object is represented as an item of the `Input` object class.

`Hidden` objects don't respond to any events. They can't since they are not visible and therefore the user cannot interact with them to trigger one.

They may be accessed by event handling functions associated with the `Form` object but very little else.

Unlike MSIE, Netscape does not support the `defaultValue` property for this sub-class of the `Input` object. It also does not support the `onFocus` event, although why MSIE should support focus control onto a `Hidden` object is not clear.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="RESULT">?</DIV>
<FORM>
   <INPUT TYPE="hidden" VALUE="A" NAME="BOX_A">
   <INPUT TYPE="hidden" VALUE="B" NAME="BOX_B">
   <INPUT TYPE="hidden" VALUE="C" NAME="BOX_C">
   <INPUT TYPE="hidden" VALUE="D" NAME="BOX_D">
   <INPUT TYPE="button" VALUE="Reveal" onClick="handleClick()">
</FORM>
<SCRIPT>
function handleClick()
{
   myString  = "[";
   myString += document.forms[0].BOX_A.value;
   myString += "] [";
   myString += document.forms[0].BOX_B.value;
   myString += "] [";
   myString += document.forms[0].BOX_C.value;
   myString += "] [";
   myString += document.forms[0].BOX_D.value;
   myString += "]";
   document.all.RESULT.innerText = myString;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Element` object, `Form.elements[]`, `Input` object

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	Warning
`onRowEnter`	-	3.0 +	-	4.0 +	-	-	-
`onRowExit`	-	3.0 +	-	4.0 +	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

Hidden.type (Property)

The `type` value for the `<INPUT>` object that describes the hidden field.

Availability:	DOM level - 1 JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myHidden`*`.type`

The `type` value for a hidden field is always "`hidden`". This value is necessary to determine the type of form element because this object is really an instance of the `Input` class.

See also:	`Input.type`

Property attributes:

`ReadOnly`.

Hidden.value (Property)

The `value` of a hidden field in a form.

Availability:	DOM level - 1 JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera browser - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myHidden`*`.value`

The value of a hidden form is probably its main purpose for existence. The idea is that you can pass values into a form by means of hidden fields and when the form is submitted, those values are passed back to the server with the form data that the user has entered. It is often used as a means of maintaining state across a user session.

See also:	`Input.value`

Hiding scripts from old browsers (Pitfall)

Old browsers need to have script content disguised.

If you are presenting your HTML on the web to an audience that may use browsers that cannot cope with the `<SCRIPT>` tag, then you will need to hide the script inside the `<SCRIPT>` block. This is done by placing HTML comment tags around the script code.

This depends on the JavaScript interpreter providing some non-standardized behavior. That is, it needs to recognize the HTML comment opening tag as being some valid syntax, even though it is not really legal.

Beware that the closing HTML comment tag is placed on a line that itself is a JavaScript comment. However, you must also make sure that the closing `</SCRIPT>` tag is placed on a line of its own or the interpreter cannot see it and the `<SCRIPT>` block then becomes unterminated.

Example code:

```
<SCRIPT>
<!-- // Hide from older browsersalert("some script here");// end hiding -->
</SCRIPT>
```

See also:	`COMMENT` object, Pitfalls

Cross-references:

Wrox *Instant JavaScript* - page 46

Hierarchy of objects (Definition)

To fully understand JavaScript and in particular its use in the browser, it is helpful to know how objects relate to one another.

In the core JavaScript language, objects are related to one another by means of the prototype chain. This is a hierarchy that determines the inheritance of functionality from a super-class, although it's an object-based inheritance and not really a class-based inheritance. You can only walk up this hierarchy. There is no convenient way to determine what objects are sub-classed from an object.

From version 5.0 of MSIE and version 6.0 of Netscape, there are several hierarchy models available for traversing the document. The HTML view of the document is not quite the same as the DOM view. In general, each one is based on a property to traverse up the tree to a parent and collections for traversing down the tree to the leaf nodes. Here are some examples of the different hierarchical arrangements.

- The `parentElement` and `children[]` collection operate on the HTML tag-based hierarchy.
- The `parentNode` property and `childNodes[]` collection can be used to traverse the DOM hierarchy which exposes the interstitial text objects between the HTML tags. These are not visible to the HTML view.
- The spatial layout and positioning of objects relative to one another can be walked upwards by means of the `offsetParent` property. You can't traverse this tree from top to bottom without inspecting the contents of the `children[]` collection and looking at each child's `offsetX` and `offsetY` properties.

- The `all[]` collection flattens the whole tree from the receiving object downwards. The `document.all[]` collection is the complete tree. This can slow things down if you are searching it to find a single object.
- An editing hierarchy is constructed with the `parentTextEdit` property which describes the relationship between items that can have a text range created for them, and hence have some selectable content that can be cut or copied to the clipboard (usually as text).
- Function calls construct a hierarchy by means of the `Argument` objects which link upwards to the calling function's `Arguments` object. This provides a way to walk through a stack trace to diagnose a calling sequence.
- You can create custom hierarchies by using nested `<DIV>` tags and navigating them by means of the corresponding `DIV` objects they instantiate.
- A spatial hierarchy is created in Netscape version 4 with the `<LAYER>` tags. Layers sharing a common parent are siblings and can be ordered relative to one another.
- A CSS style hierarchy is implemented in MSIE to describe the relationship between CSS style rule objects and their owning parent style sheet.
- Another hierarchy can be built by importing style sheets into one another to allow styles to be managed in a more modular fashion. This is also supported at the rule level by means of the `styleSheet.addRule()` method.

The DOM level 2 standard adds a suite of traversal methods for walking through a document object model. It is embodied in the following classes:

- `NodeIterator`
- `NodeFilter`
- `TreeWalker`
- `DocumentTraversal`

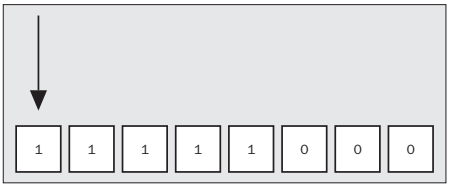
Related to the traversal suite, the DOM level 2 standard also introduces the Range suite. This is embodied in these classes:

- `Range`
- `DocumentRange`
- `RangeException`

See also: `Arguments.caller`, `ChildNodes` object, `DIV` object, `Document.createTextNode()`, `Element.all[]`, `Element.childNodes[]`, `Element.children[]`, `Element.offsetParent`, `Element.ownerDocument`, `Element.parentElement`, `Element.parentNode`, `Element.parentTextEdit`, `Layer.siblingAbove`, Prototype Based Inheritance, `rule.parentStyleSheet`, `StyleSheet.addImport()`, `StyleSheet.addRule()`, `StyleSheet.owningElement`, `StyleSheet.owningNode`, `StyleSheet.parentStyleSheet`

High order bit (Definition)

The most significant bit in an integer value.



See also:	Bit, Bit-field, Bitwise operator, byte

History (Background)

Scripting language history.

Availability:	ECMAScript edition - 2

The JavaScript language was invented by Brendan Eich at Netscape. In the early days, it was known as LiveScript.

The ECMA 262 standard is based on the earlier work at Netscape where they embedded the JavaScript interpreter into the Netscape version 2.0 web browser. Microsoft also embedded script-driven capabilities into their version 3.0 of the MSIE browser. The Microsoft interpreter implements a language called JScript as opposed to JavaScript.

Microsoft clearly chose not to use the word JavaScript. It might be because any product name containing the word 'Java' needs to be licensed from Sun Microsystems or that it was a Netscape originated name. Given the very public antagonism between Microsoft and the other two, this is understandable. Perhaps it also gives Microsoft a little extra leeway to extend the language in non-standard ways.

Both Microsoft and Netscape have continued to enhance their implementations in subsequent versions of their web browsers, taking them sometimes in completely opposing directions.

In November 1996, the language started to become standardized by a working group under the ECMA organization. At this stage it was also commonly referred to as ECMAScript and became a published standard in June 1997. By April 1998, the ECMA 262 standard had been adopted as an international standard as ISO/IEC 16262 which prompted a second edition of the ECMA standard to keep the two fully aligned.

A third edition of the ECMA 262 standard was published in October 1999.

On the horizon are upgraded interpreters from Netscape and Microsoft, and the availability of JavaScript in a variety of other products such as TV set-top boxes, mobile phones and embeddings into legacy applications.

See also:	JavaScript language, JavaScript version, JScript version

Cross-references:

ECMA 262 edition 2 - section Introduction

ECMA 262 edition 3 - section Introduction

Wrox *Instant JavaScript* - page 3

history (Property)

An alias to the `window.history` property.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`History object`	
JavaScript syntax:	-	`history`
	-	*`myWindow`*`.history`

Property attributes:

`ReadOnly`.

Refer to:

`Window.history`

History object (Object/browser)

A `history` object owned by the `window`. This exposes information about URLs that have been visited previously.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
JavaScript syntax:	-	*`myHistory`* `= history`
	-	*`myHistory`* `=` *`myWindow`*`.history`
Object properties:	`current, length, next, previous`	
Object methods:	`back(), forward(), go()`	

Netscape version 4 provides access to the `history` array by signed scripts. Earlier versions of Netscape and MSIE do not provide this level of access and therefore the `history` object is limited in what you can do with it.

Some properties can be accessed by non-privileged scripts.

In Netscape, each element in the `History` object array is a String containing the URL for that item in the history. In MSIE, the objects are not accessible directly.

Warnings:

- The array elements and properties of this object cannot be accessed by JavaScript unless the script has the `UniversalBrowserRead` privilege granted to it.
- On Netscape, the `toString()` method is not correctly implemented and returns the value "[`object`]" instead of "[`object History`]".

See also:	`Collection` object, `UniversalBrowserAccess`, `UniversalBrowserRead`, `Window.history`

Property	**JavaScript**	**JScript**	**N**	**IE**	**Opera**	**HTML**	**Notes**
`current`	1.1 +	-	3.0 +	-	3.0 +	-	Warning, ReadOnly.
`length`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	ReadOnly.
`next`	1.1 +	-	3.0 +	-	3.0 +	-	Warning, ReadOnly.
`previous`	1.1 +	-	3.0 +	-	3.0 +	-	Warning, ReadOnly.

Method	**JavaScript**	**JScript**	**N**	**IE**	**Opera**	**HTML**	**Notes**
`back()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-
`forward()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-
`go()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	Warning

History.back() (Method)

Go to the previous page.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
JavaScript syntax:	-	*`myHistory`*`.back()`

This is accessible to unsigned scripts.

See also:	`Window.back()`

History.current (Property)

The URL of the current window content.

Availability:	JavaScript - 1.1 Netscape Navigator - 3.0 Opera browser - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myHistory*.current

If you are accessing the URL of the current page, it is probably better to use the `location` object and access its `href` property.

Warnings:

- The `UniversalBrowserRead` privilege is required to access this property value.

See also:	`UniversalBrowserRead`

Property attributes:

`ReadOnly`.

History.forward() (Method)

Go to the next page.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
JavaScript syntax:	-	*myHistory*.forward()

This is accessible to unsigned scripts.

See also:	`Window.forward()`

History.go() (Method)

Return to a URL from the `history` array.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera browser - 3.0	
JavaScript syntax:	-	*myHistory*.go(*anIndex*)
	-	*myHistory*.go(*aURL*)
	-	*myHistory*.go(*aDocTitle*)
Argument list:	*anIndex*	Denotes which history item to go to
	aURL	A URL value to go to
	aDocTitle	The name of a document (in Netscape)

You can perform a soft reload of a page, retaining its current settings by means of the `History.go(0)` method call.

The following values are meaningful as an argument:

- -4 indicates you want to go back 4 pages.
- -1 indicates you want to go back one page.
- 0 requests a reload of the current page
- 2 indicates you want to go forward two pages
- A URL indicates a specific page to load
- A Document title references the history list associatively

Warnings:

- There are some bugs in this method in versions of Netscape prior to version 4. MSIE version 3 also exhibits unruly behavior.

Example code:

```
// Recall a page from the history keeping form settings intact
history.go(-1);
```

See also:	`Location.reload()`

History.length (Property)

The number of history entries available.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myHistory`*`.length`

Property attributes:

`ReadOnly`.

Refer to:

`Collection.length`

History.next (Property)

The next document in the `history` array.

Availability:	JavaScript - 1.1 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*`myHistory`*`.next`

The value you get back from this property will depend very much on what navigation the user has recently performed. Most of the time this would yield a `null` or undefined value unless the user has used the [BACK] button.

Warnings:

- The `UniversalBrowserRead` privilege is required to access this property value.
- This property is not supported on the WebTV platform.

See also:	JellyScript, `UniversalPreferencesRead`

Property attributes:

`ReadOnly`.

History.previous (Property)

The previous document in the `history` array.

Availability:	JavaScript - 1.1 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myHistory*.previous

You may find it more useful to build a stack of pages visited in a persistent array. This can be tricky within a single window, but if you are prepared to use a frameset, you can create a global session store for such things. That way you'd be able to traverse a logical history rather than the possibly random page ordering that the user navigated.

Warnings:

- The `UniversalBrowserRead` privilege is required to access this property value.

See also:	`UniversalPreferencesRead`

Property attributes:

`ReadOnly`.

H<n> object (Object/HTML)

An object that represents the `<H1>` to `<H6>` tags.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myH1* = *myDocument*.all.*anElementID*
	IE	*myH1* = *myDocument*.all.tags("H1")[*anIndex*]
	IE	*myH1* = *myDocument*.all[*aName*]
	-	*myH1* = *myDocument*.getElementById(*anElementID*)
	-	*myH1* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myH1* = *myDocument*.getElementsByTagName("H1")[*anIndex*]

HTML syntax:	`<H1> ... </H1>,<H2> ... </H2>,<H3> ... </H3>,<H4> ... </H4>,<H5> ... </H5>,<H6> ... </H6>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align`	
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	

The properties, methods, collections and event handling support for the `<H1>` tag object are also provided for `<H2>` through `<H6>` tag objects as well. It is only necessary to document one of these object classes, although the other header types are each instantiated as objects of an appropriately named class in MSIE. The syntax examples illustrate the use of an `<H1>` tag object.

The `<H1>` through `<H6>` tags are block-level tags. That means that they force a line break before and after themselves.

The DOM level 1 specification refers to this and its sibling object types as a `HeadingElement` object.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

H<n>.align (Property)

The alignment of an <H1> to <H6> tag.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myH1*.align

The alignment of the H1 (to H6) object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- absbottom
- absmiddle
- baseline
- bottom
- center
- left
- middle
- right
- texttop
- top

home() (Method)

Go to the home page according to the user preferences.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`home()`
	N	*myWindow*.home()

Refer to:

Window.home()

Host environment (Definition)

The environment in which a JavaScript interpreter has been embedded.

The host environment is where the JavaScript interpreter lives.

Typically this might be a web browser, a web server back end, or an embedded implementation. There are other possibilities as well, limited only by the imagination of the software developers and their ingenuity in finding new ways to apply JavaScript.

JavaScript is usually interpreted. However, recent digital TV systems are deploying byte code compiled JavaScript and it is also being used in that way for mobile devices.

See also:	`<SCRIPT>`, `argc` parameter, `argv` parameter, CGI Driven JavaScript, Desktop JavaScript, Embedded JavaScript, Environment, Execution environment, File extensions, Glue code, Host features, Implementation-defined behavior, `main()` function, PDF, Platform, Script execution, Shell Scripting with JavaScript, Web browser, WScript

Cross-references:

Wrox *Instant JavaScript* - page 5

Wrox *Instant JavaScript* - page 42

Host features (Definition)

That which is added to the language due to the hosting environment it runs in.

Host features are those which the implementation adds around the native feature set. Host features would include new object types. There are entire groups of objects which collected together provide a host environment capability.

For example, the HTML Document Object Model describes a page as it lives inside a web browser.

The PDF file format can contain a collection of objects and the PDF reader adds others to construct a different kind of object model.

A process control system might contain an object model representation of a brewery or a nuclear reactor, although it is unlikely you would control either with just a JavaScript interpreter. You could build a monitoring system with a mimic display that was a fairly good illustration of the processes though.

The browser, server or operating system is said to host the interpreter. That is where the terminology of host environments, host features and so on is derived.

See also:	ECMAScript, Host environment

Cross-references:

Wrox *Instant JavaScript* - page 12

Wrox *Instant JavaScript* - page 42

Host object (Definition)

An object that is built-in, but is provided by the hosting environment.

Availability:	ECMAScript edition - 2

A host object is any additional non-native object provided by the interpreter at the outset of script execution.

Host objects may or may not strictly honor the intent of the `Get` and `Put` internal function. The ECMA standard allows for the possibility that a host object may still properly manage `Get` and `Put` internal functions, even if the `HasProperty` function returns `false` for the properties being accessed.

See also:	`argv` parameter, `for( ... in ... ) ...`, `HasProperty()`, Implementation-defined behavior, `main()` function

Cross-references:

ECMA 262 edition 2 - section 4.3.8

ECMA 262 edition 3 - section 4.3.8

HR object (Object/HTML)

An object that represents an `<HR>` tag.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myHR* = *myDocument*.all.*anElementID*
	IE	*myHR* = *myDocument*.all.tags("HR")[*anIndex*]
	IE	*myHR* = *myDocument*.all[*aName*]
	-	*myHR* = *myDocument*.getElementById(*anElementID*)
	-	*myHR* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myHR* = *myDocument*.getElementsByTagName("HR")[*anIndex*]
HTML syntax:	`<HR>`	

Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align`, `color`, `noShade`, `size`, `width`	
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	

The `<HR>` tag is a block-level tag. That means that it forces a line break before and after itself.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`color`	-	3.0 +	-	4.0 +	-	-	-	Warning
`noShade`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`size`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`width`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

HR.align (Property)

An attribute to control the alignment of the <HR> object on screen.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myHR*.align

The alignment of the HR object with respect to its containing parent object is defined in this property. The following expected and widely available set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

HR.color (Property)

The color of the <HR> object.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myHR*.color

The color of the horizontal rule defined by this HR object will be defined in this property.

The color can be specified in the normal way according to the HTML color specifiers.

Warnings:

- In JavaScript the `noShade` and `color` properties operate independently of one another, even though setting the `COLOR="..."` HTML tag attribute implies a `NOSHADE` attribute as well when defined in the document HTML source.

See also:	Color names, Color value

HR.noShade (Property)

A switch attribute, used to control whether there is a shadow around the`<HR>` object.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myHR*`.noShade`

This Boolean value can be set to `true` which will change the horizontalrule style from the 3D appearance to a flat non-shaded look. Setting the property to `false` restores the 3D-shaded appearance again.

Warnings:

- In JavaScript the `noShade` and `color` properties operate independently of one another even though setting the `COLOR="..."` HTML tag attribute implies a `NOSHADE` attribute as well when defined in the document HTML source.

HR.size (Property)

A measure of the thickness of the `<HR>` object, expressed in pixels.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myHR*`.size`

The thickness of the horizontal rule, measured in pixels.

HR.width (Property)

A width measurement of the <HR> object.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myHR*.width

The width of the rule can be defined in pixels as an absolute measure or by specifying the value as a percentage of its containing element's width.

Although the value is noted as being a String primitive, a numeric value is specified for the width measured in pixels. It is a string value simply sot hat it can accommodate the percentage value.

HTC (Definition)

This is an abbreviation for the MSIE based HTML components (which used to be called scriptlets).

See also:	Scriptlet, HTML Component

.htc (File extension)

A file containing a behavior handler script for use in MSIE browsers.

This file contains an HTML Component (HTC). These used to be called scriptlets but have been evolved and renamed when used in an MSIE web browser.The Windows Script Host environment continues to use scriptlets but these have also evolved into something quite different to an HTC.

Refer to the `Element.addBehavior()` topic for a more detailed explanation of how these files are used.

See also:	`<STYLE>, Element.addBehavior(), Scriptlet`

.htm (File extension)

An HTML page.

Refer to:

File extensions

HTML (Standard)

The standard notation for creating web pages.

HTML is short for HyperText Markup Language. It is a markup language belonging to the SGML (Standard Generalized Markup Language) family.

Currently, the HTML state of the art is framed in the HTML version 4.0 standard as defined by the World Wide Web Consortium (W3C). However, much web content is still presented in an HTML 3.2 conformant manner.

TV set-top boxes that employ HTML are generally built around an HTML 3.2 core which means that web content that needs to be deployed to the PC based browser and the TV viewer needs to be downgraded to be compatible with that standard version.

Browser manufacturers are still yet to release fully HTML 4.0 compliant browsers.

The strategic importance of HTML version 4.0 is to separate document content from presentation style. This means that certain techniques are deprecated in favor of the use of style sheets.

This adds much complexity to the JavaScript support since its API to those style controls is not as well standardized as the underlying Document Object Model (DOM). However, there is also significant flexibility in the new styling model, and to date, the MSIE browser has made great strides in implementing a robust styling model that provides many Dynamic HTML capabilities.

It is hoped that the forthcoming Netscape 6.0 might offer similar capabilities.

See also:	Web browser

.html (File extension)

An HTML page.

Refer to:

File extensions

HTML Character entity (Definition)

Character value escapes for use in HTML.

HTML Character entities are escape sequences that work in the HTML source domain. The character entities have a defined format that begins with an ampersand as a control sequence introducer and terminates with a semi-colon. The value between them is the character description.

Many of these have symbolic names which browsers honor to a greater or lesser extent depending on the browser and its vintage. HTML 4.0 defines a large number of these escaped entities. If the character you wish to represent does not have a symbolic name, you can use the decimal value of the character and then place a hash symbol in-front. You can also use the hexadecimal notation by placing an x in front of the value. Thus:

`<` defines the less-than or < symbol.

`'` defines a single quote or apostrophe character.

`'` also defines a single quote character.

Here is a table of useful character entity values:

Numeric	Named	Description	Netscape	MSIE
&#001;	-	-	Missing	Space
&#002;	-	-	Missing	Space
&#003;	-	-	Missing	Space
&#004;	-	-	Missing	Space
&#005;	-	-	Missing	Space
&#006;	-	-	Missing	Space
&#007;	-	-	Invisible	Space
&#008;	-	-	Missing	Space
	-	Horizontal tab	Space	Space
-	Line feed	Space	Space
&#011;	-	-	Space	Space
	-	-	Space	Space
-	Carriage return	Space	Space
&#014;	-	-	Missing	Space
&#015;	-	-	Missing	Space
&#016;	-	-	Missing	Space
&#017;	-	-	Missing	Space
&#018;	-	-	Missing	Space
&#019;	-	-	Missing	Space
&#020;	-	-	Missing	Space
&#021;	-	-	Missing	Space

Table continued on following page

Numeric	Named	Description	Netscape	MSIE
&#022;	-	-	Missing	Space
&#023;	-	-	Missing	Space
&#024;	-	-	Missing	Space
&#025;	-	-	Missing	Space
&#026;	-	-	Missing	Space
&#027;	-	-	Missing	Space
&#028;	-	-	Missing	Space
&#029;	-	-	Missing	Space
&#030;	-	-	Missing	Space
&#031;	-	-	Missing	Space
&#032;	-	Space	Space	Space
&#033;	-	Exclamation mark	Yes	Yes
&#034;	&quot;	Double quote	Yes	Yes
&#035;	-	Hash	Yes	Yes
&#036;	-	Dollar	Yes	Yes
&#037;	-	Percent	Yes	Yes
&#038;	&amp;	Ampersand	Yes	Yes
&#039;	-	Apostrophe	Yes	Yes
&#040;	-	Left parenthesis	Yes	Yes
&#041;	-	Right parenthesis	Yes	Yes
&#042;	-	Asterisk	Yes	Yes
&#043;	-	Plus	Yes	Yes
&#044;	-	Comma	Yes	Yes
&#045;	-	Hyphen	Yes	Yes
&#046;	-	Period	Yes	Yes
&#047;	-	Slash	Yes	Yes
&#048;	-	Digit 0	Yes	Yes
&#049;	-	Digit 1	Yes	Yes
&#050;	-	Digit 2	Yes	Yes
&#051;	-	Digit 3	Yes	Yes
&#052;	-	Digit 4	Yes	Yes
&#053;	-	Digit 5	Yes	Yes
&#054;	-	Digit 6	Yes	Yes
&#055;	-	Digit 7	Yes	Yes
&#056;	-	Digit 8	Yes	Yes
&#057;	-	Digit 9	Yes	Yes
&#058;	-	Colon	Yes	Yes

Table continued on following page

Numeric	Named	Description	Netscape	MSIE
&#059;	-	Semicolon	Yes	Yes
&#060;	&lt;	Less than	Yes	Yes
&#061;	-	Equals	Yes	Yes
&#062;	&gt;	Greater than	Yes	Yes
&#063;	-	Question mark	Yes	Yes
&#064;	-	Commercial at sign	Yes	Yes
&#065;	-	Letter A	Yes	Yes
&#066;	-	Letter B	Yes	Yes
&#067;	-	Letter C	Yes	Yes
&#068;	-	Letter D	Yes	Yes
&#069;	-	Letter E	Yes	Yes
&#070;	-	Letter F	Yes	Yes
&#071;	-	Letter G	Yes	Yes
&#072;	-	Letter H	Yes	Yes
&#073;	-	Letter I	Yes	Yes
&#074;	-	Letter J	Yes	Yes
&#075;	-	Letter K	Yes	Yes
&#076;	-	Letter L	Yes	Yes
&#077;	-	Letter M	Yes	Yes
&#078;	-	Letter N	Yes	Yes
&#079;	-	Letter O	Yes	Yes
&#080;	-	Letter P	Yes	Yes
&#081;	-	Letter Q	Yes	Yes
&#082;	-	Letter R	Yes	Yes
&#083;	-	Letter S	Yes	Yes
&#084;	-	Letter T	Yes	Yes
&#085;	-	Letter U	Yes	Yes
&#086;	-	Letter V	Yes	Yes
&#087;	-	Letter W	Yes	Yes
&#088;	-	Letter X	Yes	Yes
&#089;	-	Letter Y	Yes	Yes
&#090;	-	Letter Z	Yes	Yes
&#091;	-	Left square bracket	Yes	Yes
&#092;	-	Backslash	Yes	Yes
&#093;	-	Right square bracket	Yes	Yes
&#094;	-	Caret	Yes	Yes

Table continued on following page

Numeric	Named	Description	Netscape	MSIE
&#095;	-	Underscore	Yes	Yes
&#096;	-	Grave accent	Yes	Yes
&#097;	-	Letter a	Yes	Yes
&#098;	-	Letter b	Yes	Yes
&#099;	-	Letter c	Yes	Yes
&#100;	-	Letter d	Yes	Yes
&#101;	-	Letter e	Yes	Yes
&#102;	-	Letter f	Yes	Yes
&#103;	-	Letter g	Yes	Yes
&#104;	-	Letter h	Yes	Yes
&#105;	-	Letter i	Yes	Yes
&#106;	-	Letter j	Yes	Yes
&#107;	-	Letter k	Yes	Yes
&#108;	-	Letter l	Yes	Yes
&#109;	-	Letter m	Yes	Yes
&#110;	-	Letter n	Yes	Yes
&#111;	-	Letter o	Yes	Yes
&#112;	-	Letter p	Yes	Yes
&#113;	-	Letter q	Yes	Yes
&#114;	-	Letter r	Yes	Yes
&#115;	-	Letter s	Yes	Yes
&#116;	-	Letter t	Yes	Yes
&#117;	-	Letter u	Yes	Yes
&#118;	-	Letter v	Yes	Yes
&#119;	-	Letter w	Yes	Yes
&#120;	-	Letter x	Yes	Yes
&#121;	-	Letter y	Yes	Yes
&#122;	-	Letter z	Yes	Yes
&#123;	-	Left curly brace	Yes	Yes
&#124;	-	Vertical bar	Yes	Yes
&#125;	-	Right curly brace	Yes	Yes
&#126;	-	Tilde	Yes	Yes
&#127;	-	Undefined	Displays ?	Missing
&#128;	-	Undefined	No	Displays ?
&#129;	-	Undefined	No	Displays ?
&#130;	-	Comma	Yes	Yes
&#131;	-	Florin	Yes	Yes

Table continued on following page

Numeric	Named	Description	Netscape	MSIE
&#132;	-	Right double quote	Yes	Yes
&#133;	-	Ellipsis	Yes	Yes
&#134;	-	Dagger	Yes	Yes
&#135;	-	Double dagger	Yes	Yes
&#136;	-	Circumflex	Yes	Yes
&#137;	-	Permil	Yes	Yes
&#138;	-	Undefined	Displays ?	Bugged
&#139;	-	Less than	Yes	Yes
&#140;	-	Capital OE ligature	Yes	Yes
&#141;	-	Undefined	No	Displays ?
&#142;	-	Undefined	No	Displays ?
&#143;	-	Undefined	No	Displays ?
&#144;	-	Undefined	No	Displays ?
&#145;	-	Left single quote	Yes	Yes
&#146;	-	Right single quote	Yes	Yes
&#147;	-	Left double quote	Yes	Yes
&#148;	-	Right double quote	Yes	Yes
&#149;	-	Bullet	Yes	Yes
&#150;	-	En dash	Yes	Yes
&#151;	-	Em dash	Yes	Yes
&#152;	-	Tilde	Yes	Yes
&#153;	-	Trademark symbol	Yes	Yes
&#154;	-	Undefined	Displays ?	Yes
&#155;	-	Greater than	Yes	Yes
&#156;	-	Small oe ligature	Yes	Yes
&#157;	-	Undefined	No	Displays ?
&#158;	-	Undefined	No	Displays ?
&#159;	-	Capital Y umlaut	Yes	Yes
		Non breaking space	Yes	Yes
&#161;	&iexcl;	Inverted exclamation	Yes	Yes
&#162;	&cent;	Cent sign	Yes	Yes
&#163;	&pound;	Pound sign	Yes	Yes
&#164;	&curren;	General currency symbol	Displays ?	Displays ?
&#165;	&yen;	Yen sign	Yes	Yes
&#166;	&brvbar;	Broken vertical bar	Displays ?	Yes
&#167;	&sect;	Section sign	Yes	Yes
&#168;	&uml;	Umlaut	Yes	Yes

Table continued on following page

Numeric	Named	Description	Netscape	MSIE
&#169;	&copy;	Copyright	Yes	Yes
&#170;	&ordf;	Feminine ordinal	Yes	Yes
&#171;	&laquo;	Left angle quote	Yes	Yes
&#172;	&not;	Not sign	Yes	Yes
&#173;	&shy;	Soft hyphen	Displays ?	Yes
&#174;	&reg;	Registered trademark	Yes	Yes
&#175;	&macr;	Macron accent	Yes	Yes
&#176;	&deg;	Degree sign	Yes	Yes
&#177;	&plusmn;	Plus or minus	Yes	Yes
&#178;	&sup2;	Superscripted 2	Displays ?	Yes
&#179;	&sup3;	Superscripted 3	Displays ?	Yes
&#180;	&acute;	Acute accent	Yes	Yes
&#181;	&micro;	Micro sign	Yes	Yes
&#182;	&para;	Paragraph	Yes	Yes
&#183;	&middot;	Middle dot	Yes	Yes
&#184;	&cedil;	Cedilla	Yes	Yes
&#185;	&sup1;	Superscripted 1	Displays ?	Yes
&#186;	&ordm;	Masculine ordinal	Yes	Yes
&#187;	&raquo;	Right angle quote	Yes	Yes
&#188;	&frac14;	One quarter	Displays ?	Yes
&#189;	&frac12;	One half	Displays ?	Yes
&#190;	&frac34;	Three quarters	Displays ?	Yes
&#191;	&iquest;	Inverted question mark	Yes	Yes
&#192;	&Agrave;	Capital A grave	Yes	Yes
&#193;	&Aacute;	Capital A acute	Yes	Yes
&#194;	&Acirc;	Capital A circumflex	Yes	Yes
&#195;	&Atilde;	Capital A tilde	Yes	Yes
&#196;	&Auml;	Capital A umlaut	Yes	Yes
&#197;	&Aring;	Capital A ring	Yes	Yes
&#198;	&AElig;	Capital AE ligature	Yes	Yes
&#199;	&Ccedil;	Capital C cedilla	Yes	Yes
&#200;	&Egrave;	Capital E grave	Yes	Yes
&#201;	&Eacute;	Capital E acute	Yes	Yes
&#202;	&Ecirc;	Capital E circumflex	Yes	Yes

Table continued on following page

Numeric	Named	Description	Netscape	MSIE
&#203;	&Euml;	Capital E umlaut	Yes	Yes
&#204;	&Igrave;	Capital I grave	Yes	Yes
&#205;	&Iacute;	Capital I acute	Yes	Yes
&#206;	&Icirc;	Capital I circumflex	Yes	Yes
&#207;	&Iuml;	Capital I umlaut	Yes	Yes
&#208;	&ETH;	Capital eth	Displays ?	Bugged
&#209;	&Ntilde;	Capital N tilde	Yes	Yes
&#210;	&Ograve;	Capital O grave	Yes	Yes
&#211;	&Oacute;	Capital O acute	Yes	Yes
&#212;	&Ocirc;	Capital O circumflex	Yes	Yes
&#213;	&Otilde;	Capital O tilde	Yes	Yes
&#214;	&Ouml;	Capital O umlaut	Yes	Yes
&#215;	&times;	Multiply	Displays ?	Yes
&#216;	&Oslash;	Capital O slash	Yes	Yes
&#217;	&Ugrave;	Capital U grave	Yes	Yes
&#218;	&Uacute;	Capital U acute	Yes	Yes
&#219;	&Ucirc;	Capital U circumflex	Yes	Yes
&#220;	&Uuml;	Capital U umlaut	Yes	Yes
&#221;	&Yacute;	Capital Y acute	Displays ?	Yes
&#222;	&THORN;	Capital thorn	Displays ?	Bugged
&#223;	&szlig;	Small sz ligature	Yes	Yes
&#224;	&agrave;	Small a grave	Yes	Yes
&#225;	&aacute;	Small a acute	Yes	Yes
&#226;	&acirc;	Small a circumflex	Yes	Yes
&#227;	&atilde;	Small a tilde	Yes	Yes
&#228;	&auml;	Small a umlaut	Yes	Yes
&#229;	&aring;	Small a ring	Yes	Yes
&#230;	&aelig;	Small ae ligature	Yes	Yes
&#231;	&ccedil;	Small c cedilla	Yes	Yes
&#232;	&egrave;	Small e grave	Yes	Yes
&#233;	&eacute;	Small e acute	Yes	Yes
&#234;	&ecirc;	Small e circumflex	Yes	Yes
&#235;	&euml;	Small e umlaut	Yes	Yes
&#236;	&igrave;	Small i grave	Yes	Yes
&#237;	&iacute;	Small i acute	Yes	Yes
&#238;	&icirc;	Small i circumflex	Yes	Yes

Table continued on following page

Numeric	Named	Description	Netscape	MSIE
&#239;	&iuml;	Small i umlaut	Yes	Yes
&#240;	&eth;	Small eth	Displays ?	Bugged
&#241;	&ntilde;	Small n tilde	Yes	Yes
&#242;	&ograve;	Small o grave	Yes	Yes
&#243;	&oacute;	Small o acute	Yes	Yes
&#244;	&ocirc;	Small o circumflex	Yes	Yes
&#245;	&otilde;	Small o tilde	Yes	Yes
&#246;	&ouml;	Small o umlaut	Yes	Yes
&#247;	&divide;	Divide	Yes	Yes
&#248;	&oslash;	Small o slash	Yes	Yes
&#249;	&ugrave;	Small u grave	Yes	Yes
&#250;	&uacute;	Small u acute	Yes	Yes
&#251;	&ucirc;	Small u circumflex	Yes	Yes
&#252;	&uuml;	Small u umlaut	Yes	Yes
&#253;	&yacute;	Small y acute	Displays ?	Yes
&#254;	&thorn;	Small thorn	Displays ?	Bugged
&#255;	&yuml;	Small y umlaut	Yes	Yes

Warnings:

- Inside `<SCRIPT>` tags, you are inside the JavaScript source domain and you use a different set of escape mechanisms. If you use the HTML escape mechanisms inside JavaScript source, your script is likely to break unless you are intentionally outputting them to the HTML source space via a `document.write()` method.
- The character set you are using inside the `<SCRIPT>` tags is completely different to that used in HTML. The script source is written using Unicode characters (ISO 10646). If you are generating HTML, a great manyof the character codes are mapped differently. This is because the HTML character entities are defined according to an ISO standard (ISO 8859). Even then, national language variants of the browser should map the character entity values to the correct character glyph which should ensure that what the designer intended is what you actually see, regardless of the actual character code transformations that take place.
- Most of the character entities are supported by both MSIE and Netscape. A few characters are supported inconsistently. All of the character entities with values less than 32 are control codes anyway, so although the browsers treat them differently, it shouldn't prove to be significant. Those character entities that are undefined in the HTML specification aren't supported in the same way either, and should probably be avoided. Recent updates to the HTML standard allow for Unicode character values to be used above character entity 255, and some of these have symbolic names defined. Refer tothe HTML specification available from the W3C at http://www.w3.org/TR/html4/ for further details.
- Of the rest:
- The general currency symbol (¤) is unsupported.

- There is currently no Euro symbol implemented although its character entity •and Unicode value 20AC are defined but not in the version of the Unicode standard that is currently mandated (version 2.0). Unicode is about to undergo a revision and at that time, browser manufacturers will then need to accommodate the changes to remain compliant.
- The broken vertical bar ( ) does not work on Netscape, nor does the soft hyphen (), and neither do the superscripted numbers and fractional values.
- ETH and THORN are broken in both browsers in both upper and lower case variants.
- Netscape also lacks support for Y acute in both upper and lower case.
- Most of these are fairly obscure and not likely to cause much difficulty aside from some very specialized national language support. TheEuro character, however, is likely to become more important whether the currency thrives or not.

See also:	Escaped JavaScript quotes in HTML,Portability

Web-references:

http://www.w3.org/TR/html4/

HTML Comment tag (<!– ... –>) (HTML Tag)

HTML comments can be used to hide scripts.

See also:	COMMENT object, Hiding scripts from old browsers

HTML Component (Definition)

An HTML Component is a small modular fragment of script contained in an `.htc` file and can be shared across several pages.

These used to be called scriptlets but have evolved into HTML Components. Do not confuse them with Windows Script Host scriptlets: these are something different altogether.

See also:	Scriptlet

Web-references:

http://msdn.microsoft.com/workshop

HTML entity escape (Pitfall)

It looks like HTML but it isn't intended to be.

You can sometimes innocently include some text into your script that when presented with a`document.write()`, gets completely misunderstood by the HTML parser. This will almost certainly be due to the presence of "<" and">" characters in the output. It is likely that the browser will see what it thinks is a tag, but then ignore it according to the "I don't know what it is - so I won't display it" rule, as it won't be a recognized tag.

Use HTML escapes to output the character as intended.

This is important for the following characters if not for others:

```
< becomes &lt;

> becomes &gt;

& becomes &amp;
```

See also:	Pitfalls

Cross-references:

Wrox *Instant JavaScript* - page 46

HTML file (Definition)

HTML web page file.

JavaScript is contained in HTML files with the `<SCRIPT>` tag. This is called client side JavaScript.

See also:	`<SCRIPT>`, File extensions, Web browser

Cross-references:

Wrox *Instant JavaScript* –page 3

HTML object (Object/HTML)

An object in MSIE that represents an `<HTML>` tag.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 3.0 Internet Explorer - 4.0 Netscape - 6.0

Inherits from:	`Element` object	
JavaScript syntax:	IE	*myHTML* = *myDocument*.all.*anElementID*
	IE	*myHTML* = *myDocument*.all.tags("HTML")[0]
	IE	*myHTML* = *myDocument*.all.tags("HTML")[*anIndex*]
	IE	*myHTML* = *myDocument*.all[*aName*]
	IE	*myHTML* = *myDocument*.all[*anIndex*]
	-	*myHTML* = *myDocument*.documentElement
	-	*myHTML* =*myDocument*.getElementById(*anElementID*)
	-	*myHTML* =*myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myHTML* =*myDocument*.getElementsByTagName ("HTML")[*anIndex*]
HTML syntax:	`<HTML> ... </HTML>`	
Argument list:	*anIndex*	A selector for one particular HTML element
	aName	An associative array reference
	anElementID	The ID value of an `Element` object
Object properties:	`title,version`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

This object is otherwise known as the HTML object.

The `<HTML>` tag is a block-level tag, although it cannot be placed inside any other tag. Like the body tag, it is considered to be a block-level tag on grounds of its behavior in a framed context.

Warnings:

- Be careful how you operate on this object. Traversing its properties in a `for( ... in ... )` loop can recursively lock up your browser.
- The index value that points at this object in the `all[]` array for the document will be 0 if there is no DTD statement and 1 if there is. A correctly formed document should have a DTD statement. Use a dynamic mechanism for locating the object of type HTML instead of hardwiring the index value.

See also:	`Attributes` object, `BODY.aLink`, `BODY.background`, `BODY.bgColor`, `BODY.link`, `BODY.text`,`BODY.vLink`, `Document` object, `Document.documentElement`, `Document.title`, `Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`title`	-	3.0 +	-	4.0 +	-	-	-	-
`version`	1.5+	5.0 +	6.0 +	5.0 +	-	1 +	-	Warning , Deprecated

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

HTML.title (Property)

The document title.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myHTML*`.title`
HTML syntax:	`<TITLE>`*aTitle*`</TITLE>`	
Argument list:	*aTitle*	A title text

The DOM standard at level 1 requires that the title property of an HTML object should contain the text from the `<TITLE>` HTML tag in the`<HEAD>` block of the document. At least, this is the implication, because it refers the reader to the HTML specification, which defines that `<TITLE>`is only legal inside `<HEAD>` blocks. We can't really say this is part of the DOM specification, which unfortunately makes it ambiguous.

Netscape makes this available as the `document.title` property, as does MSIE. However, MSIE does support this property as a member of the HTML object even though it is not mentioned in the Microsoft documentation.

In Netscape, the title property of an HTML object exists but is empty, hence it cannot be guaranteed to be available in Netscape 6.0.

See also:	`Document.title`

HTML.version (Property)

The version string of an HTML document.

Availability:	DOM level - 1 JavaScript - 1.5 JScript - 5.0 Internet Explorer - 5.0 Netscape - 6.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myHTML*.version
HTML syntax:	<HTML VERSION="*aVersion*">	
Argument list:	*aVersion*	A version text

This property is intended to hold the value of the VERSION="..." HTML tag attribute which is used in conjunction with the <HTML> tag. It has now been superseded by the <!DOCTYPE> DTD tag and is now deprecated.

The MSIE browser does reflect this HTML tag into the version property, although it is not mentioned in the documentation.

Warnings:

- This feature is deprecated in favor of placing aproperty DTD statement at the start of the document.

HTML tag attribute (Definition)

HTML tags have attributes, some of which can be accessed fromscripts.

Earlier versions of the browsers required that HTML tag attributes were enclosed in double quotes. More recent browsers also allow the use of single quotes or even no quotes in some cases. This may be useful when enclosing fragments of JavaScript that contain quoted text strings.

See also:	Attribute object, Attributes object

HTTP-EQUIV="..." (HTML Tag Attribute)

The name attribute for a pseudo header item.

Refer to:

<META>

http: URL (Request method)

A request from a web browser to a web server to send a document.

This requests a document from a web server. Most web traffic isrequested this way.

See also:	javascript: URL, URL

https: URL (Request method)

A request from a web browser to a secure web server to send a document with an encrypted and secure protocol.

Requests a document from a secure web server. Your encryption code needs to be compatible and this may involve an exchange of security certificates.

See also:	javascript: URL, Security policy, URL

HyperLink (Definition)

HyperLinks are references to documents served elsewhere. They may be pages, assets or includable files.

See also:	`Anchor.href, BASE.href,IMG.href, Location.href, StyleSheet.href,` HTML

HyperLink object (Object/HTML)

Another name for the `Url` object.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0	
Inherits from:	`Element` object	
JavaScript syntax:	-	*myHyperLink* = *myDocument*.links[*anIndex*]
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

In Netscape, links are stored in `Url` objects. These are distinctly different to `Anchor` objects.

MSIE does not distinguish between the two but since there is no constructor, it is hard to know what object type they are. Generally they are assumed to be `Url` objects.

Because the class name is `Url` in Netscape, the link objects are discussed in detail under that lexical topic location.

MSIE supports a `LINK` object class but this is a special object that stems from a `styleSheet` item. It doesn't support all the properties that a `Url` object does and is probably more concerned with managing the appearance of a `Url` object on the screen.

See also:	`Area` object, `Element`.all[], `LinkArrayobject`, URL, `Urlobject`

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

I object (Object/HTML)

An object that represents the font style controlled by the <I> HTML tag.

Availability:	DOM level –1 JScript –3.0 Internet Explorer –4.0 Deprecated	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myI* = *myDocument*.all.*anElementID*
	IE	*myI* = *myDocument*.all.tags("I")*[anIndex]*
	IE	*myI* = *myDocument*.all[*aName*]
	-	*myI* = *myDocument*.getElementById(*anElementID*)
	-	*myI* = *myDocument*.getElementsByName(*aName*)[anIndex]
	-	*myI* = *myDocument*.getElementsByTagName("I")[anIndex]
HTML syntax:	<I> ... </I>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

<I> tags and the objects that represent them are inline elements. Placing them into a document does not create a line break.

See also:	`Element` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

iCab (Web browser)

A web browser alternative to MSIE and Netscape Navigator.

The iCab web browser is only available on the Macintosh but has some interesting features not found in other products. It is developed by a small German company run by Alexander Clauss and Oliver Joppich. The browser was originally developed for use on the Atari platform.

The browser is compact, fast and standards-based and can be downloaded free from the iCab web site (http://www.iCab.de). The current version is 2.1 and is available at the moment as a preview or Beta product (see web-references).

Earlier versions do not fully support JavaScript although implementation of a completely standards based version of the interpreter is expected in the final release. It also supports some of the browser specific tags that Netscape and Microsoft have added to their browsers. This is necessary to avoid the browser not being able to properly render the pages currently deployed on the web. However, iCab also has an HTML validator built in and this triggers an indicator on the screen when the page is non-HTML 4.0 compliant. This shows up as a smiley face when the HTML is good and a frowning face when it's not. Clicking on the frowning face yields a report of the non-compliant HTML. This is a good browser for developers to use.

iCab supports some interesting features such as image and cookie filtering. This allows much finer control over image display than simply switching images on and off. By default, iCab is set up to not filter out banner ads, but by virtue of its awareness of standard banner ad image sizes it can prevent the display of banners leaving other graphics intact. It can also filter based on URL contents. Cookie filtering is also flexible and sophisticated.

The `<LINK REL="...">` and `<LINK REV="...">` tags are actively supported and provide some structural navigation when sites implement these tags properly.

Access the iCab web site for more details of special features.

See also:	Platform, Script execution, Web browser

Web-references:

http://www.icab.de/

ID="..." (HTML Tag Attribute)

MSIE document objects can be referenced conveniently with an ID name if it is added with this tag attribute.

Many objects are identified in the DOM hierarchy of the web browser by means of their id property. This value is defined as an HTML tag attribute.

Some objects can be accessed using a `<NAME="...">` HTML tag attribute instead of or as well as the `ID="..."` HTML tag attribute.

Netscape 4.0 supports ID access but only apparently for visible elements. Version 6.0 is much more consistent now that it supports the DOM standard properly.

See also:	`Document.ids[],Document.layers[], Document.scripts[],Element.id,NAME="..."`

Identically equal to (===) (Operator/identity)

Compares two values for equality and identical type.

Availability:	ECMAScript edition –3 JavaScript –1.3 JScript –1.0 Internet Explorer –3.02 Netscape –4.06	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*anOperand1 === anOperand2*
Argument list:	*anOperand1*	A value of a comparable type
	anOperand2	A value of the same type as operand 1

The two operands are compared and the Boolean `true` value is returned if they are equal and of the same type.

The equality operator is not always transitive. For example, two string objects may represent the same string value. Comparing the objects for equality will yield `false` because the references to the objects are being compared and not the object values themselves. However forcing a string comparison may in fact yield a `true` value when testing for equality. Do this by converting the objects as part of the comparison process by type conversion or `valueOf()` methods.

Numeric values may require rounding to take place, and testing for equality may be accurate down to a precision finer than your script cares about. You can set a precision value in a variable, then subtract the value you are comparing with from the absolute value of the comparee. If the difference is smaller than your precision value, then the two values are close enough to yield a match.

The associativity is from left to right.

Refer to the operator precedence topic for details of execution order.

The result is the Boolean value `true` if *`anOperand1`* is numerically or lexically equal to *`anOperand2`* and both operands are of the same type otherwise `false` is returned.

Warnings:

- Be careful not to confuse the single and double equals operators with the triple equals operator.
- Placing a single equals in place of a test for equality will assign the right-hand value to the left-hand operand and clobber the value accidentally. Placing a single equals sign in-stead of the identity operator has the same effect.
- Using the equality operator in place of the identity operator is more subtle. Sometimes the test will appear to work correctly because the values in the two objects could be the same. That would have failed the identity test because they may be equal but are not identical.
- The interpreter may be forgiving enough that a run-time error isn't generated, but the side effects could be subtle and make it hard to diagnose the cause.
- This is not available for use server-side with Netscape Enterprise Server 3.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myObject1 = 100;
myObject2 = "100";
if(myObject1 == myObject2)
{
   document.write("Objects are equal<BR>");
}
else
{
   document.write("Objects are NOT equal<BR>");
}
```

```
if(myObject1 === myObject2)
{
   document.write("Objects are identical<BR>");
}
else
{
   document.write("Objects are NOT identical<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also: ASCII, Associativity, Equal to (==), Equality expression, Equality operator, Greater than (>), Greater than or equal to (>=), Identity operator, JellyScript, Less than (<), Less than or equal to (<=), Logical expression, Logical operator, NOT Equal to (!=), NOT Identically equal to (!==), Operator Precedence, Relational expression, Relational operator, typeof, Unicode

Cross-references:

ECMA 262 edition 3 –section –11.9.4

O'Reilly JavaScript Definitive Guide –page –48

Wrox Instant JavaScript –page –39

Identifier (Definition)

A means of referring uniquely to a variable, property or method.

Variables and functions have unique names. These names are called identifiers.

An identifier is a string of characters, composed of letters, digits or two other special characters ($ and _). An identifier name must begin with a non-digit (that is a letter or underscore). JavaScript is case-sensitive and an identifier spelled with uppercase letters is distinct from one having the same name spelled with lowercase letters.

You should not use any of the reserved keywords as identifier names.

An identifier refers to a variable, property or function and is the user-defined way to distinguish one item from another.

Identifiers can be of unlimited length, although for practical purposes they should be kept to a reasonable length, say 30 to 40 characters at most. The following characters are valid for use in identifiers:

a b c d e f g h i j k l m

n o p q r s t u v w x y z

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

$ _ (Underscore)

Identifiers are case-sensitive. The following all denote different identifiers:

```
exampleidentifier

exampleIdentifier

ExampleIdentifier

EXAMPLEidentifier

EXAMPLEIDENTIFIER
```

The dollar sign should be used sparingly and according to the ECMA standard is reserved for use by mechanically generated code.

Warnings:

- You cannot use reserved words as identifier names.
- Avoid using the dollar sign. MSIE version 3.0 and Netscape 2.02 cannot cope with dollar signs in variable names. MSIE version 3.02 cannot tell the difference between one identifier and another when the names only differ in the case of the letters that spell the identifier name.

See also:	Digit, `function( ... ) ...`, `int`, Internet Explorer, Lexical element, Namespace, Netscape Navigator, Nondigit, Scope chain, Storage duration, String literal, Token, Type, `with ...`

Cross-references:

ECMA 262 edition 2 –section –7.5

ECMA 262 edition 3 –section –7.6

O'Reilly *JavaScript Definitive Guide* –page –31

Wrox *Instant JavaScript* –page –14

Identifier resolution (Definition)

The process of resolving and locating an identifier.

Availability:	ECMAScript edition –2

Identifiers are resolved by binding the name to the value container through the scope chain.

A hierarchical access mechanism adds items to the front of the scope chain as code is nested. The most local variable object belonging to the current execution context is searched first. If the name is not resolved, then the scope chain is walked until a matching name is located or the global code's execution context is reached. If there is still no match, then the undefined value is returned.

An identifier is considered to be a primary expression when it is being evaluated.

See also:	Binding, Completion type, Execution context, Primary expression, Scope chain, `with ...`

Cross-references:

ECMA 262 edition 2 –section –10.1.4

ECMA 262 edition 2 –section –11.1.2

ECMA 262 edition 3 –section –10.1.4

ECMA 262 edition 3 –section –11.1.2

Identity operator (Definition)

A close cousin of the equality operator family.

Equality operators are generally used for comparing primitive values.

If they are used for comparing objects, the objects are converted to primitives and then tested. This may not be the test you are trying to accomplish.

Imagine two objects, each distinct from one another but containing the same value in the properties that are yielded by the `toString()` or `valueOf()` methods. They are different objects but they will test `true` for equality.

The identity operators provide a way to test whether they are the same object or not.

There are two identity operators:

- === Identically equal to
- !=== Not identically equal to

You can use these in relational expressions in much the same way as you would the equality operators but remember that you are testing references to objects not values.

See also:	Identically equal to (===), NOT Identically equal to (!==)

ids (Property)

An alternative reference to the `document.ids` property in JSS.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Collection object`	
JavaScript syntax:	N	`ids`
	N	*`myDocument`*`.ids`

Warnings:

- This functionality is removed from Netscape 6.0.

See also:	JavaScript Style Sheets, Document.ids[]

IEEE 754 (Standard)

An international standard for floating point number handling and storage in 8 bytes.

Availability:	ECMAScript edition –2

The IEEE 754 standard defines the behavior of a numeric environment in such a way that the computation should generate the same result across any compliant platforms.

It specifies the exact format for the storage and manipulation of the values. It also specifies bounding ranges for exponents and mantissas.

The standards describes how and when rounding should occur and the direction in which rounding takes place. Exceptions are also described and this determines how the `NaN` value is generated and propagated through expressions.

See also:	byte, Floating constant, `NaN`, Not a number, Number

Cross-references:

ECMA 262 edition 2 –section –3

ECMA 262 edition 3 –section –3

if(...) ... (Selector)

A conditional execution mechanism.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
JavaScript syntax:	-	*aLabel*: if(*aCondition*) { *someCode*}
Argument list:	*aCondition*	An expression that yields a Boolean value
	aLabel	An optional identifier to name the if block
	someCode	Some script source text that is executed if the condition tests `true`

The `if()` statement is used to conditionally execute a code block depending on the result of a conditional expression. This mechanism is called branching.

The expression in the parentheses is evaluated and cast to a Boolean value. If it yields a `true` value as a result, then the code in the associated block is executed. Otherwise, the code is ignored and execution continues at the line following the closing brace of the code block.

JavaScript allows you to omit the braces around the code block if the code is a single line. You must place the trailing semi-colon on the line to delimit the script source text that is associated with the `if()` statement.

At version 1.2 of JavaScript, you can name the `if` statement and use the labelled form of the break keyword to exit the conditional code block prematurely.

Warnings:

- Beware of leaving the braces off the associated script source text as it is possible for this to be ambiguous to the reader and can lead to difficulties when adding more code to be executed conditionally. It can become easy to add a second line of code but still omit the braces. In that case, only the first line will be conditional but the second will always be executed regardless of the result of the `if()` condition. This can lead to completely unexpected behavior and is quite difficult to diagnose.

Example code:

```
// Reccomended form
if(a == b)
{
   z = 100;
   alert("z set to 100");
}

// Possibly dangerous during maintenance
if(a !== b)
   z = 100;
```

See also: Code block delimiter `{}`, Compound statement, Conditionally execute (`?:`), `else ...`, `else if( ... ) ...`, Flow control, `if( ... ) ... else ...`, Obfuscation, Selection statement, Statement, `switch( ... ) ... case: ... default: ...`, `while( ... ) ...`

Cross-references:

ECMA 262 edition 2 –section –12.5

ECMA 262 edition 2 –section –12.6.1

ECMA 262 edition 3 –section –12.5

Wrox *Instant JavaScript* –page –22

if(...) ... else ... (Selector)

A conditional execution mechanism.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
JavaScript syntax:	-	*aLabel*: if(*aCondition*) { *someCode*} else { *someOtherCode*)
Argument list:	*aCondition*	An expression that yields a Boolean value
	aLabel	An optional identifier to name the if block
	someCode	Some script source text that is executed `if` the condition tests `true`
	someOtherCode	Some script source text that is executed if the condition tests `false`

The `if() ... else` statement is used to conditionally execute one or other code block depending on the result of a conditional expression.

The expression in the parentheses is evaluated and cast to a Boolean value. If it yields a `true` value as a result, then the code in the first associated block is executed. Otherwise, the code in the first block is ignored and the code in the block following the `else` keyword is executed.

Each `else` keyword for which the associated `if()` is ambiguous will be associated with the nearest possible `if()` that would otherwise have no corresponding `else`.

At version 1.2 of JavaScript, you can name the `if` statement and use the labelled form of the break keyword to exit the conditional code block prematurely.

Example code:

```
// Reccomended form
if(a == b)
{
   z = 100;
}
else
{
   z = 200;
}

// Possibly dangerous during maintenance
if(a == b)
z = 100;
else
z = 200;
```

See also:	Code block delimiter `{}`, Compound statement, Conditionally execute (`?:`), `else ...`, `else if( ... ) ...`, Flow control, `if( ... ) ...`, Selection statement, Statement, `switch( ... ) ... case: ... default: ...`

Cross-references:

ECMA 262 edition 2 –section –12.5

ECMA 262 edition 3 –section –12.5

Wrox *Instant JavaScript* –page –22

IFRAME object (Object/HTML)

An object that represents an <IFRAME> tag.

Availability:	DOM level –1 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myIFRAME* = *myDocument*.all.*anElementID*
	IE	*myIFRAME* = *myDocument*.all.tags("IFRAME")[*anIndex*]
	IE	*myIFRAME* = *myDocument*.all[*aName*]
	-	*myIFRAME* = *myDocument*.*aName*
	-	*myIFRAME* = *myDocument*.getElementById(*anElementID*)
	-	*myIFRAME* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myIFRAME* = *myDocument*.getElementsByTagName("IFRAME")[anIndex]
HTML syntax:	`<IFRAME> ... </IFRAME>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align, dataFld, dataSrc, frameBorder, frameSpacing, height, hspace, longDesc, marginHeight, marginWidth, name, noResize, scrolling, src, tabIndex, vspace, width`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

An `<IFRAME>` is a special MSIE supported tag that introduces an inline frame into a document that appears like a block structured element within the document text flow, accessible as a named frame object that belongs to the document.

Its properties appear to be all read-only in the MSIE browser. Changing them seems to have no visible effect at all. The properties belonging to the object retain the values that you assign and return them when requested but the display does not change.

The DOM level 1 specification refers to `IFRAME` objects as `IFrameElement` objects and this makes them available in Netscape by virtue of its support for DOM level 1.

DOM level 2 adds the following properties:

- `contentDocument`

Warnings:

- Netscape does not support the `<IFRAME>` tag prior to version 6.0, but it does support an `<ILAYER>` tag which describes an inline laye,r which is not the same thing but may provide a way to emulate the `<IFRAME>` functionality in some cases.
- MSIE seems to have trouble locating an `IFRAME` object in the document hierarchy, and you cannot refer to the object directly in the same way that you can with other objects.
- With an `IFRAME` object whose ID is "MYFRAME" this accessor works:
  - `document.all.MYFRAME`
- But these accessors don't:
  - `document.MYFRAME`
  - `MYFRAME`
- Even worse, the IFRAME appears to float in some separate plane to its containing object. Placing an `<IFRAME>` inside a `<DIV>` block exhibits some very strange behavior. Its position seems locked to the top left of the `<DIV>` but its right edge seems to be able to flow outside the `<DIV>` area.
- Netscape 6.0, which implements `IFRAME` objects, does not appear to fare any better, exhibiting similarly strange behavior.

See also:	Element object, Frame object, Window object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`dataFld`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`dataSrc`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`frameBorder`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`frameSpacing`	-	3.0 +	-	4.0 +	-	-	-	Warning
`height`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`hspace`	-	3.0 +	-	4.0 +	-	-	-	Warning
`longDesc`	1.5 +	-	6.0 +	-	-	1 +	-	Warning
`marginHeight`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`marginWidth`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
name	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
noResize	-	3.0 +	-	4.0 +	-	-	-	-
scrolling	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
src	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
vspace	-	3.0 +	-	4.0 +	-	-	-	Warning
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5+	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5+	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5+	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5+	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

IFRAME.align (Property)

A controlling attribute that affects the alignment of the `<IFRAME>` with respect to its parent object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*.align

The alignment of the `IFRAME` object with respect to its containing parent object is defined in this property. An expected and widely available set of alignment specifiers are available, as follows (the default is bottom):

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

Warnings:

- None of the alignment values seem to affect the appearance of the `IFRAME` object in MSIE or Netscape 6.0.

IFRAME.frameBorder (Property)

Describes a 3D border effect to be drawn round the inline frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*`.frameBorder`

The `frameBorder` property should allow you to turn the engraved frame border effect on and off. Its should accept the following values (the default is yes):

- 0
- 1
- yes
- no

Warnings:

- This seems to have no effect on the appearance of the `IFRAME` object in the MSIE or Netscape browsers.

See also:	`Frame.frameBorder`

IFRAME.frameSpacing (Property)

A frame spacing margin around the `<IFRAME>` object.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myIFRAME*`.frameSpacing`

A spacing distance between multiple frames that is measured in pixels on the screen.

Warnings:

- The spacing controls for an IFRAME are provided in differently named properties and HTML tag attributes to those supported by the normal `<FRAME>` in `<FRAMESET>` objects.
- They don't seem to work on the MSIE or Netscape browsers at all.

See also:	`Frame.frameBorder`, `Frame.marginHeight`, `Frame.marginWidth`

IFRAME.height (Property)

Contains the height of the inline frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*`.height`

The current height of the IFRAME can be measured from this property. You ought to be able to modify it but it seems not to work.

Warnings:

- This appears to be a read-only property in MSIE and Netscape.

See also:	`Frame.height`, `IFRAME.width`

IFRAME.hspace (Property)

A measure of the horizontal spacing either side of the `<IFRAME>` to separate it from any adjacent objects.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myIFRAME*`.hspace`

This works the same as the `hspace` you apply to an `image` object. It introduces some space between objects that run together on the same line. It appears as if the margin has been altered but it is a different spacing control.

The space is applied equally on both sides of the `IFRAME` object.

Warnings:

- This has no visible effect on the `IFRAME` object in MSIE and Netscape.

See also:	`IFRAME.vspace`

IFRAME.longDesc (Property)

This is a URL which points at a document containing a long description of the contents of this frame.

Availability:	DOM level –1 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myIFRAME*`.longDesc`

It needs to be defined as the `IFRAME` is created to have any meaningful purpose.

Warnings:

- This yields nothing in MSIE and Netscape.

See also:	`Frame.longDesc`

IFRAME.marginHeight (Property)

A measure of the vertical margin above and below the `<IFRAME>` object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*`.marginHeight`

Margins flow round the entire frame. You cannot operate on all four individually but you can operate on the vertical and horizontal margin pairs independently.

That means the top and bottom margin must be the same, as must the left and right.

This property is the size of the margins at the top and bottom of the frame.

This is a bit like the `vspace` property but since neither seems to have any effect it is difficult to prove they operate independently.

Warnings:

- This seems to have no visible effect in MSIE and Netscape.

See also:	`Frame.marginHeight`

IFRAME.marginWidth (Property)

A measure of the horizontal margin to the left and right of the `<IFRAME>` object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*`.marginWidth`

Margins flow round the entire frame. You cannot operate on them individually but you can operate on the vertical and horizontal margins separately.

That means the top and bottom margin must be the same, as must the left and right.

This property is the size of the margins to the left and right of the frame.

This is a bit like the `hspace` property but since neither seems to have any effect it is difficult to prove that they operate independently.

Warnings:

- This seems to have no visible effect in MSIE and Netscape.

See also:	`Frame.marginWidth`

IFRAME.name (Property)

The value of the `NAME="..."` HTML tag attribute.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myI*`FRAME.name`

This reflects the value defined in the HTML tag attribute. It's not likely that you would want to change it.

See also:	`Window.name`

IFRAME.noResize (Property)

A switch value that controls whether the `<IFRAME>` can be resized or not.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myI*`FRAME.noResize`

This is a Boolean value to turn the frame resizing control on and off. Unlike the border control, this is a truly Boolean value accepting either `true` or `false` as its setting.

However, it has no effect whatsoever –this is because inline frames can't be resized in Internet Explorer 4.0 anyway.

See also:	`Frame.noResize`

IFRAME.scrolling (Property)

A switching attribute that controls the appearance of scrollbars on the `<IFRAME>` when its content exceeds the available space.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*`.scrolling`

Yet another deviant property in the browser mess! Having just discussed the `noResize` property which is a switching mechanism having a Boolean setting, here we are with another switching property. However, as is the case with border controls, this one is not Boolean. Instead it accepts the values:

- `yes`
- `no`
- `auto`

See also:	`Frame.scrolling`

IFRAME.src (Property)

The URL of the document content inside the `<IFRAME>` object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*`.src`

Inline frames have their content loaded in much the same way as for a window. This property describes the source URL of the document that controls the content of an inline frame. You can reload the inline frame by redefining this value, but it is probably better to use the `location.href` value for that.

See also:	`Frame.src`

IFRAME.tabIndex (Property)

A numeric ordering of the <IFRAME> within the parent document's tabbing sequence.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myI*FRAME.tabIndex

This value indicates where in the tabbing sequence this object and any of its children will be placed. The tabbing order is used when filling in forms. Pressing the [tab] key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

IFRAME.vspace (Property)

A spacing in the vertical axis between the <IFRAME> and its adjacent objects.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	Number primitive	
JavaScript syntax:	IE	*myI*FRAME.vspace

This works the same as the vspace you apply to an image object. It introduces some space between objects that run together on adjacent lines. It appears as if the margin has been altered but it is a different spacing control.

The space is applied equally above and below the IFRAME object.

Warnings:

- Changing this seems to have no visible effect on the IFRAME object in MSIE and Netscape.

See also:	IFRAME.hspace

IFRAME.width (Property)

Contains the width of the inline frame.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIFRAME*`.width`

The current width of the IFRAME can be measured from this property. You ought to be able to modify it, but it doesn't seem to work.

Warnings:

- This appears to be a read-only property in MSIE and Netscape.

See also:	`IFRAME.height`

IIS (Product)

Internet Information Server. A Microsoft server product.

Refer to:

Internet Information Server

Image animation (Useful tip)

Using cached images to animate an image.

If you cache some images in local memory, you can perform some very slick animations.

You can effect animation with the `lowsrc` property in the same way as you can with the `src` property but the `src` property must remain empty while you do that. If you specify both and the browser is running in a low resolution device, the `lowsrc` image will be used instead of the `src` image.

The example shows you how to do animation with `src` properties.

Example code:

```
<HTML>
<BODY>
<!-- Image cache technique -->
<SCRIPT>
// First create an image buffer array
var myImages = new Array(10);

// Now load the array with image objects getting them from the server
for(var myIndex = 0; myIndex<10; myIndex++)
{
   myImages[myIndex] = new Image();
   myImages[myIndex].src = "assets/image_" + myIndex + ".jpg";
}

var mySequence = 0;
function animate()
{
   mySequence = (mySequence + 1)%10;
   document.images[0].src = myImages[mySequence].src;
}

setInterval("animate()", 100);
</SCRIPT>

<IMG SRC="assets/image_0.jpg" NAME="targetimage">

</BODY>
</HTML>
```

See also:	`Image.lowsrc`, `Image.src`

Image object (Object/HTML)

An object representing an HTML `<IMG>` tag in Netscape.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myImage* = *myDocument*.all.*anElementID*
	IE	*myImage* = *myDocument*.all.tags("IMG")[*anIndex*]
	IE	*myImage* = *myDocument*.all[*aName*]
	-	*myImage* = *myDocument*.*anImageName*
	-	*myImage* = *myDocument*.getElementById(*anElementID*)
	-	*myImage* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myImage* = *myDocument*.images[*anIndex*]
	-	*myImage* = *myI*mageArray[*anIndex*]
	-	*myImage* = *myDocument*.getElementsByTagName("IMG")[*anIndex*]

HTML syntax:	`<IMG>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`border, complete, constructor, defaultValue, height, hspace, lowsrc, name, size, src, vspace, width, x, y`	
Object methods:	`select()`	
Event handlers:	`onAbort, onBlur, onClick, onDblClick, onError, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

The `image` object added at version 1.1 of JavaScript introduced the possibility of dynamically replacing images under script control.

Event handling support via properties containing function objects was added to `Image` objects at version 1.1 of JavaScript.

The `image` object supported by Netscape and the `IMG` object supported by MSIE are so different to each other that they are covered as separate objects. They share a few similarities but not many. By inspection, they are instances of different classes.

Warnings:

- In the MSIE browser, the images are instantiated inside `IMG` objects because they correspond to the `IMG` tag. There is an `Image` object created as another name for the same `IMG` tag and if you instantiate a new copy of it you get an `IMG` object. This means you should be careful when examining the constructors and class names of `image` objects, as their object type is not portable across platforms.

Example code:

```
<HTML>
<HEAD>
<SCRIPT>
function moved()
{
   document.all.ONE.width = event.x;
}
</SCRIPT>

</HEAD>

<BODY onMouseMove="moved()">
Move mouse horizontally to scale image<BR>

<IMG HEIGHT=40 ID="ONE" SRC="assets/image_9.jpg">

</BODY>
</HTML>
```

See also:	Background object, Document.images[], Element.all[], ImageArray object, IMG object, Input object, Input.type, Web browser

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
border	1.1 +	-	3.0 +	-	3.0 +	1 +	-	-
complete	1.1 +	-	3.0 +	-	3.0 +	-	-	Warning, ReadOnly
constructor	1.1 +	-	3.0 +	-	-	-	-	-
defaultValue	1.1 +	-	3.0 +	-	3.0 +	1 +	-	Warning
height	1.1 +	-	3.0 +	-	3.0 +	1 +	-	ReadOnly
hspace	1.1 +	-	3.0 +	-	3.0 +	1 +	-	ReadOnly
lowsrc	1.1 +	-	3.0 +	-	3.0 +	1 +	-	-
name	1.1 +	-	3.0 +	-	3.0 +	1 +	-	-
size	1.1 +	-	3.0 +	-	3.0 +	1 +	-	Warning
src	1.1 +	-	3.0 +	-	3.0 +	1 +	-	-
vspace	1.1 +	-	3.0 +	-	3.0 +	1 +	-	ReadOnly
width	1.1 +	-	3.0 +	-	3.0 +	1 +	-	ReadOnly
x	1.1 +	-	3.0 +	-	-	-	-	ReadOnly
y	1.1 +	-	3.0 +	-	-	-	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
select()	1.1 +	-	3.0 +	-	3.0 +	1 +	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAbort	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	-	-
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onClick	1.1+	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onError	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onFocus	1.1+	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onLoad	1.1+	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.1+	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

Image() (Constructor)

A constructor for new browser images.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
JavaScript syntax:	-	`new Image()`

When the Image constructor is used in a new expression it creates a new object based on the `Image` prototype.

You can then assign values to the properties of that new `Image` object as needed.

`Image` objects are generally constructed so that image assets can be requested from the server and retained in the browser cache. This can happen in the background without the image needing to appear on the display.

Then, when you need to perform an image replacement, the browser knows that the image is stored locally and can be located more quickly than if it had to request it from the server at the time it is required.

This can significantly speed up the animation effects you choose to reproduce.

See also:	Web browser

Image() (Function)

A function for constructing new browser images.

Availability:	JavaScript –1.1 Netscape –3.0	
JavaScript syntax:	N	`Image()`

You can either use the function to create new images or another `image` object which can be cloned instead.

See also:	`Image()`, Web browser

Image.border (Property)

The thickness of the border round an image.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*`myImage.border`*

The border will be highlighted if the image is embedded inside an anchor tag.

Image.Class (Property/internal)

Internal property that returns an object class.

This is an internal property that describes the class that an `Image` object instance is a member of. The reserved words suggest that in the future, this property may be externalized.

See also:	Web browser

Property attributes:

`DontEnum`, `Internal`.

Image.complete (Property)

The current state of an image loading operation.

Availability:	JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*`myImage.complete`*

This is a Boolean property that reflects the current loading status of an `image` object. It should yield the value `false` until the image has completely transferred from the server to the client browser.

Warnings:

- Netscape 4 will incorrectly yield a `true` value when the image has not yet completely loaded.

Property attributes:

`ReadOnly`.

Image.constructor (Property)

An `Image` object constructor.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`Image object`	
JavaScript syntax:	N	*`myImage.constructor`*

The constructor is that of the built-in `Image` prototype object.

You can use this as one way of creating arrays, although it is more popular to use the new `Image()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

Image.height (Property)

The height of an image.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myImage.height`*

The image space is defined by an extent rectangle that surrounds the space occupied by it on the screen. The extent rectangle is the smallest rectangle that can completely enclose the item. This property specifies the height of that extent rectangle.

Including height and width information on images is optional, but it can significantly improve the performance of the layout engine as it renders the web page. This is because the layout engine does not need to wait until the image has been fetched before reserving sufficient space for it in the display.

See also:	`Image.width`, `IMG.height`, `IMG.width`

Property attributes:

`ReadOnly`.

Image.hspace (Property)

The horizontal spacing attribute value.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myImage.hspace`*

Margins placed around objects are either modified separately with all four margin sides having a different property or by adjusting the horizontal margins and vertical margins using just two values.

The `hspace` property controls the margin to the left and right of the object.

See also:	IMG.hspace

Property attributes:

`ReadOnly.`

Image.lowsrc (Property)

The low-resolution version of the image can be supplied from this URL.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*`myImage.lowsrc`*

This is a useful technique to make the site appear to be faster than it really is. You specify a low resolution version of the image that can be contained in a much smaller file. This is then loaded first by the browser while it downloads the larger image.

Although this property is read/write, changing it does not force the image to be reloaded again.

DOM level 1 describes this as a `lowSrc` property. Note the capitalization.

For this to work as intended, you must specify the `lowsrc` (`lowSrc`) property value prior to the `src` property value otherwise the high quality image will be fetched first.

See also:	Image animation, IMG.lowsrc

Image.name (Property)

This corresponds to the NAME attribute of the <IMG> tag.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myImage.name*
HTML syntax:	<IMG NAME="...">	

Objects are identified either by the NAME="..." HTML tag attribute or by the ID="..." HTML tag attribute.

Netscape shows a marginal preference for the name property, while MSIE seems slightly better disposed towards the ID property. However, in many cases both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

This property cannot be used with Image objects that are manufactured at run-time from the Image() constructor. They can be collected in an array and accessed associatively as named array elements, but images you create in script aren't part of the document and therefore can't be addressed as if they are.

See also:	IMG.name

Image.src (Property)

The URL where the image is located.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myImage.src*

If you change the src property of an image it will be replaced on the browser's display screen. The only limitation is that the images that are replaced are exactly the same shape and size as the previous image.

Some browsers may forgive you if you display differently sized images, and some may scale the images to fit.

This can lead to all manner of creative effects such as rollover highlights, clocks, animations, progress bars etc. You can also simulate checkboxes graphically as well as all manner of other useful and 'pretty' user interface widgets.

For animations to work smoothly, you should ensure that the images have been cached locally first otherwise the animation will appear somewhat jerky while they are fetched from a server.

See also:	Image animation, Image preloading, `IMG.src`

Image.vspace (Property)

The vertical spacing attribute value.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myImage.vspace`*

Margins placed around objects are either modified separately, with all four margin sides having a different property, or by adjusting the horizontal margins and vertical margins using just two values.

The `vspace` property controls the margin at the top and bottom of the object.

See also:	`IMG.vspace`

Property attributes:

`ReadOnly`.

Image.width (Property)

The width of an image.

Availability:	DOM level –1 JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myImage.width`*

The image space is defined by an extent rectangle that surrounds the space occupied by it on the screen. The extent rectangle is the smallest rectangle that can completely enclose the item. This property specifies the width of that extent rectangle.

Including height and width information on images is optional, but it can significantly improve the performance of the layout engine as it renders the web page. This is because the layout engine does not need to wait until the image has been fetched before reserving sufficient space for it in the display.

See also:	`Image.height`, `IMG.height`, `IMG.width`

Property attributes:

`ReadOnly`.

Image.x (Property)

The X coordinate of the image within the client display area.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myImage`*`.x`

The horizontal position of the object in the display, measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Property attributes:

`ReadOnly`.

Image.y (Property)

The Y coordinate of the image within the client display area.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myImage`*`.y`

The vertical position of the object in the display, measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Property attributes:

`ReadOnly`.

Image preloading (Useful tip)

A technique for caching images locally in readiness for an animation.

If you want to animate some images, you will need to cache them locally to make sure the animation moves smoothly. If you don't, the animation will be very jerky while the images are fetched from the server.

The action of setting the `src` attribute of the `image` object in the buffer array is simply to recall the images and store them in the cache. The `image` objects then act as a repository to store the `src` value so it can be assigned to the target image in an animation loop.

Warnings:

- Note that this technique is a waste of time if you set your browser cache to zero bytes capacity and force a document to be requested from the server every time, at least that is what happens in MSIE (and is what you would expect).
- Netscape adds a trick to this by caching objects that you are holding in memory and referencing via script variables –it's a little smarter, and knows that you are intentionally caching images even though you have set your cache size to zero.
- Some versions of MSIE fetch the image from the server every time you reference it, regardless of your cache settings or the fact that you have preloaded it and stored a reference to it in a variable.

Example code:

```
<!-- Image cache technique -->
<SCRIPT>
// First create an image buffer array

var myImages = new Array(10);

// Now load the array with image objects getting them from the server
for(var myIndex = 0; myIndex<10; myIndex++)
{
   myImages[myIndex] = new Image();
   myImages[myIndex].src = "assets/image_" + myIndex + ".jpg";
}

var anIndex=Math.ceil(Math.random()*10);

document.write("<IMG SRC=" + myImages[anIndex].src + " >");
</SCRIPT>
```

See also:	`Image.src`

ImageArray object (Object/browser)

A collection of `image` objects.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
JavaScript syntax:	-	*myImageArray* = *myDocument*.images
Object properties:	length	
Object methods:	item()	

The MSIE and Netscape browsers each maintain an `ImageArray` object, which is just a special case of the `Array` object. However, although they both store objects that represent images in that array, those image objects are quite different. For a start, in Netscape they are of the class "`Image`" while in MSIE they are of the class "`IMG`" (named after the HTML tag). This is fortunate in a way, because you can use this difference to detect what kind of object you are operating on if you need to perform complex image management activities.

For MSIE image objects refer to the `IMG` object topic and its properties.

For Netscape Navigator `image` objects refer to the `Image` object topic and its properties.

Warnings:

- Beware of the differences between the properties that MSIE and Netscape Navigator provide to support image management.

See also:	`Collection` object, `Document.images[]`, `Image` object, `IMG` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
length	1.1 +	3.0 +	3.0 +	4.0 +	-	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
item()	-	3.0 +	-	4.0 +	-	-	-

ImageArray.item() (Method)

An item selector for accessing a single image within the collection.

Availability:	JScript –3.0 Internet Explorer –4.0
Property/method value type:	`Image` object

JavaScript syntax:	IE	*myImageArray*.item(*anIndex*)
	IE	*myImageArray*.item(*aSelector*)
	IE	*myImageArray*.item(*aSelector*, *anIndex*)
Argument list:	*anIndex*	A zero based index into the collection
	aSelector	A textual value that selects all matching objects

Refer to:

`Collection.Item()`

ImageArray.length (Property)

A collection of image objects in the current document.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myImageArray*.length

See also:	`Image` object, `Collection.length`

Property attributes:

`ReadOnly`.

IMG object (Object/HTML)

The MSIE object wrapper for images.

Availability:	DOM level –1 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –3.0
Inherits from:	`Element` object

JavaScript syntax:	IE	*myIMG* = *myDocument*.all.*anElementID*
	IE	*myIMG* = *myDocument*.all.tags("IMG")[*anIndex*]
	IE	*myIMG* = *myDocument*.all[*aName*]
	-	*myIMG* = *myDocument*.anImageName
	-	*myIMG* = *myDocument*.getElementById(*anElementID*)
	-	*myIMG* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myIMG* = *myDocument*.images[*anIndex*]
	-	*myIMG* = *myImageArray*[*anIndex*]
	-	*myIMG* = *myDocument*.getElementsByTagName("IMG") [*anIndex*]
HTML syntax:	<IMG><INPUT TYPE="IMAGE">	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	accessKey, align, alt, border, complete, dataFld, dataFormatAs, dataSrc, defaultValue, dynsrc, fileCreatedDate, fileModifiedDate, fileSize, fileUpdatedDate, height, href, hspace, iccProfile, isMap, longDesc, loop, lowsrc, name, protocol, prototype, readyState, size, src, start, tabIndex, useMap, vspace, width	
Object methods:	select()	
Event handlers:	onAbort, onAfterUpdate, onBeforeUpdate, onBlur, onChange, onClick, onDataAvailable, onDataSetChanged, onDataSetComplete, onDblClick, onDragStart, onError, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onScroll, onSelectStart	

<IMG> tags and the objects that represent them are inline elements. Placing them into a document does not create a line break.

Event handling support via properties containing function objects was added to IMG objects at version 1.1 of JavaScript.

The DOM level 1 specification refers to this as an ImageElement object. Netscape implements an Image class with somewhat different properties. By inspecting their DOM attributes, you can see that they are instances of different classes.

See also:	Element object, Image object, ImageArray object, Input.accessKey

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
align	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
alt	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
border	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
complete	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
dataFld	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
dataFormatAs	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
dataSrc	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
defaultValue	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
dynsrc	-	3.0 +	-	4.0 +	-	-	-	Warning
fileCreatedDate	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
fileModifiedDate	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
fileSize	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
fileUpdatedDate	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
height	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
href	-	3.0 +	-	4.0 +	-	-	-	-
hspace	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
iccProfile	-	3.0 +	-	4.0 +	-	-	-	-
isMap	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
longDesc	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	Warning
loop	-	3.0 +	-	4.0 +	-	-	-	-
lowsrc	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
name	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
protocol	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
prototype	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
readyState	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
size	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
src	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
start	-	3.0 +	-	4.0+	-	2 +	-	-
tabIndex	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
useMap	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
vspace	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-
width	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
select()	1.5 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAbort	1.5 +	3.0+	3.0+	4.0 +	3.0 +	-	-	-
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onChange	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	-
onClick	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDataAvailable	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetChanged	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetComplete	-	3.0 +	-	4.0 +	-	-	-	-
onDblClick	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onError	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onLoad	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onMouseDown	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	3.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onResize	1.5 +	3.0 +	3.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

IMG.align (Property)

The alignment of an image within its surrounding objects.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*.align

The alignment of the `IMG` object with respect to its containing parent object is defined in this property. The following expected and widely available set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `none`
- `right`
- `texttop`
- `top`

IMG.alt (Property)

An alternative text used when the `Image` is defined as an input item.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*.alt

Objects can have an alternative text string associated with them. This is useful for browsers that cannot cope with the tag and hence may display the alternate text. If spoken styles are supported, the text may be read out to the user. Some browsers will also display the `alt` text as a tool-tip if the mouse pointer is positioned over the object and remains static for a few seconds.

IMG.border (Property)

Controls the border around an `image` object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*.border

The border will be highlighted if the image is embedded inside an anchor tag.

See also:	`Image.border`

IMG.complete (Property)

Returns a value indicating whether the image has completely loaded or not.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myIMG*.complete

The constructor is that of the built-in `Image` prototype object.

You can use this as one way of creating arrays, although it is more popular to use the new `Image()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	`Image.constructor`

Property attributes:

`ReadOnly`.

IMG.dynsrc (Property)

The URL of a video clip that can be played where an image would normally have been located.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myIMG*.dynsrc

This is part of the dynamic HTML model although the dynamism is inside the asset encapsulated by the object.

You can playback a video clip or traverse a VRML panorama at any location where an image would have been placed.

If you use the `DYNSRC="..."` HTML tag attribute, you should not specify the `SRC="..."` HTML tag attribute.

The loop property of the `IMG` object controls how many iterations of the animation take place.

Warnings:

- This is not even remotely portable so it is not recommended that you use this, unless your user base is captive and using a controlled release of the MSIE browser. Using a plugin video player is far more portable.

See also:	`IMG.loop`, `IMG.start`

IMG.fileCreatedDate (Property)

The date that the image file was created.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myIMG*.fileCreatedDate

Where it is possible to distinguish properties of document source files, this can tell you about the history of the document.

This read-only property described the date that an image file was created. With this you can calculate the age of the image file by subtracting that date from the current date and time.

See also:	`Document.fileCreatedDate`

Property attributes:

`ReadOnly`.

IMG.fileModifiedDate (Property)

The date that image file was last modified.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myIMG*`.fileModifiedDate`

Where it is possible to distinguish properties of image files, this can tell you about the history of the image.

This read-only property describes the date that a file was last modified. With this you can calculate the age of the file content by subtracting that date from the current date and time.

See also:	`Document.fileModifiedDate`

Property attributes:

`ReadOnly`.

IMG.fileSize (Property)

The size of the image file.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myIMG*`.fileSize`

This is the exact length of the received HTTP body portion of the image. This does not count any HTTP headers that the web server may have sent prior to the HTTP body.

On the Macintosh operating system, some image editors hide additional resource data in the file. This is not included and on that platform, the `fileSize` property is a measurement of the data fork of the file and does not include the resource fork.

See also:	`Document.fileSize`

Property attributes:

`ReadOnly`.

IMG.fileUpdatedDate (Property)

The date that the image file was last updated.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myIMG*`.fileUpdatedDate`

According to the available documentation from Microsoft, there is no distinction made between this and the `fileModifiedDate`. However this property is only available for `IMG` objects whereas the other file metrics can be obtained for `IMG` and `document` objects.

Property attributes:

`ReadOnly`.

IMG.height (Property)

The height of the image within the client display surface.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*`.height`

The image space is defined by an extent rectangle that surrounds the space occupied by it on the screen. The extent rectangle is the smallest rectangle that can completely enclose the item. This property specifies the height of that extent rectangle.

Including height and width information on images is optional but it can significantly improve the performance of the layout engine as it renders the web page. This is because the layout engine does not need to wait until the image has been fetched before reserving sufficient space for it in the display.

See also:	`Image.height`, `Image.width`, `IMG.width`

IMG.href (Property)

The URL where the image was loaded from. This is identical to the `SRC="..."` HTML tag attribute.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myIMG*`.href`

Image tags can have their HREF defined by JavaScript code. To do this, you need to place a JavaScript: URL in the HREF and return a valid location on exit from the function or expression that is invoked.

See also:	`IMG.src`

IMG.hspace (Property)

The horizontal spacing either side of the image.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*`.hspace`

Margins placed around objects are either modified separately with all four margin sides having a different property, or by adjusting the horizontal margins and vertical margins using just two values.

The `hspace` property controls the margin to the left and right of the object.

See also:	`Image.hspace`

IMG.iccProfile (Property)

The color of an image may appear different from one platform to another. This indicates the color profile that was used when the image was created so you can make adjustments if necessary.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myIMG*`.iccProfile`

Color correction software is something that should be performed on a workstation in a consistent manner across all applications. This is because they all share the same visual display, and that display needs to be calibrated to conform to a particular color temperature and gamma curve.

An ICC profile is a preset collection of values that may be accessed by symbolic name and from that you may be able to select a particular setting in the browser to ensure the images and colors in the web page are displayed exactly as the designer intended.

IMG.isMap (Property)

Indicates whether the image is acting as a server-side image map.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myIMG*`.isMap`

This Boolean value determines whether an image is used as a server-side image map. If it is a server-side image map, then a click on the image will return a URL request to the server that includes an X-Y coordinate pair describing where on the image the mouse was positioned when the button was clicked.

You can set this property yourself from a script, although it would most likely be set from HTML.

The value `true` signifies that the image is a server-side image map and the value `false` signifies that it is not.

IMG.longDesc (Property)

This is a long description of the image file.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*`.longDesc`

Warnings:

- This is a separate property to the `alt` text or the title value for the image. It does not appear to work on the Macintosh platform and only yields an empty string.

IMG.loop (Property)

This controls the looping of a video clip that is loaded with the `dynsrc` attribute.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myIMG*`.loop`

The value of this loop property of the `IMG` object controls how many iterations of the animation take place.

By setting the value to -1, continuous looping is achieved. Otherwise, the number of cycles is defined by a positive integer.

See also:	`IMG.dynsrc`

IMG.lowsrc (Property)

The URL of a low-resolution image that can be loaded quickly.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*`.lowsrc`

This is a useful technique to make the site appear to be faster than it really is. You specify a low resolution version of the image that can be contained in a much smaller file. This is then loaded first by the browser while it downloads the larger image.

Although this property is read/write, changing it does not force the image to be reloaded again.

The DOM level 1 specification refers to this as the `lowSrc` property. Note the capitalization.

For this to work as intended, you must specify the `lowsrc` (`lowSrc`) property value prior to the `src` property value otherwise the high quality image will be fetched first.

See also:	`Image.lowsrc`

IMG.name (Property)

This corresponds to the NAME attribute of the <IMG> tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –3.0 Netscape –2.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myIMG*.name

Objects are identified either by the NAME="..." HTML tag attribute or by the ID="..." HTML tag attribute.

Netscape shows a marginal preference for the name property while MSIE seems slightly better disposed towards the ID property. However, in many cases both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

This property cannot be used with IMG objects that are manufactured at run-time from the IMG() constructor. They can be collected in an array and accessed associatively as named array elements. Images you create in script aren't part of the document and therefore can't be addressed as if they are.

See also:	Image.name

IMG.protocol (Property)

The protocol that was used in the image URL.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myIMG*.protocol

This should yield the value HTTP or FILE or one of the other available protocols according to how the image was accessed. This allows you to build script code than can behave differently according to how the image was loaded.

Typically this might allow for a different page content to be displayed when an image is loaded from a local file system.

Refer to the URL topic for details of a variety of URL protocol values.

See also:	Anchor.protocol, Document.protocol, URL, Url.protocol

Property attributes:

ReadOnly.

IMG.prototype (Property)

The prototype for the `IMG` object that can be used to extend the interface for all `IMG` objects.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Image` object	
JavaScript syntax:	IE	`IMG.prototype`
	IE	*`myIMG`*`.constructor.prototype`

Property attributes:

`ReadOnly`.

Refer to:

`prototype` property

IMG.readyState (Property)

The current disposition of the image as it is being loaded.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myIMG`*`.readyState`

This property reflects the loading status of an image.

Sometimes, you can design scripts to execute while the document is downloading –inline scripts for example. At that time, you may even be able to trigger interval timed deferred executions as well.

If it is important that the document has completed loading, you can check this property for one of the following values:

State	Value
uninitialized	The object is first instantiated but has not begun loading.
loading	The object has commenced loading.
loaded	The object has completed loading.
interactive	The object is loaded but not yet closed, but is ready to handle interaction.
complete	The object body has been closed and the loading is finished.

An object may not need to reflect the complete status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an `OBJECT` object that represents an `<OBJECT>` tag until the `<BODY>` has completed loading. This is because the `ActiveX` object construction requires a complete document body structure to attach itself to.

Every time this `readyState` value changes, it triggers an `onReadyStateChange` event call-back.

See also:	`onReadyStateChange`

Property attributes:

`ReadOnly`.

IMG.src (Property)

The URL where the image can be loaded from.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*`.src`

If you change the `src` property of an image it will be replaced on the browser's display screen. The only limitation is that the images that are replaced are exactly the same shape and size as the previous image.

Some browsers may forgive you if you display differently sized images, some may scale the images to fit.

Changing the `src` property of an image can lead to all manner of creative effects such as rollover highlights, clocks, animations, progress bars etc. You can also simulate checkboxes graphically as well as all manner of other useful and 'pretty' user interface widgets.

For animations to work smoothly, you should ensure that the images have been cached locally first, otherwise the animation will appear somewhat jerky while they are fetched from a server.

See also:	`Image.src`, `IMG.href`

IMG.start (Property)

The state of the image when it started loading a dynamic source.

Availability:	DOM level –2 JScript –3.0 Internet Explorer –3.02	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myIMG*.start

This property indicates when the dynamic content should start to play. It can take two values as follows:

- `fileopen`
- `mouseover`

You should only specify this value when the IMG object has a meaningful `dynsrc` property value.

See also:	`IMG.dynsrc`

IMG.useMap (Property)

The URL of a `<MAP>` defined hash element that defines a client side image map.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*.useMap

This property reflects the value of the `USEMAP="..."` HTML tag attribute which should refer to the named `<MAP>` tag containing an image map. The reference is by means of a `"#NAME"` value in this property that corresponds to the `NAME="..."` HTML tag attribute of the `<MAP>` tag describing the image map to use.

See also:	`Map.name`

IMG.vspace (Property)

The vertical margin above and below the image.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*.vspace

Margins placed around objects are either modified separately with all four margin sides having a different property, or by adjusting the horizontal margins and vertical margins using just two values.

The `vspace` property controls the margin at the top and bottom of the object.

See also:	`Image.vspace`

IMG.width (Property)

The width of the image on the client display surface.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myIMG*.width

The image space is defined by an extent rectangle that surrounds the space occupied by it on the screen. The extent rectangle is the smallest rectangle that can completely enclose the item. This property specifies the width of that extent rectangle.

Including height and width information on images is optional but it can significantly improve the performance of the layout engine as it renders the web page. This is because the layout engine does not need to wait until the image has been fetched before reserving sufficient space for it in the display.

See also:	`Image.height`, `Image.width`, `IMG.height`

Implementation (Definition)

A JavaScript interpreter provided in a usable form that can execute scripts.

An implementation is a program or set of programs that can read, interpret and execute script source text according to the conventions of the JavaScript language.

An ECMA compliant implementation is one that does so according to the precepts laid down in the ECMA 262 standard. This would be an ECMAScript interpreter although it should also state what edition of the standard it is compliant with.

A DOM compliant interpreter would be compatible with DOM level 1 or level 2. Other levels are in the course of being standardized.

An implementation will usually supply additional functionality over and above that specified by the standard.

See also:	Definition, ECMA, ECMAScript, ECMAScript –edition 2, ECMAScript – edition 3, Interpret

Implementation object (Object/DOM)

A special core object that describes the DOM implementation.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
JavaScript syntax:	-	*myImplementation* = *myDocument*.implementation
Object methods:	hasFeature()	

This is part of the DOM compliant implementation that MSIE now provides.

It has no enumerable properties and is a member of a the `Implementation` object class. It describes the DOM implementation that is supported.

The DOM level 2 specification adds the following two methods:

- `createDocumentType()`
- `createDocument()`

Warnings:

- The DOM level 1 standard calls this a `DOMImplementation` object. MSIE calls it an `Implementation` object.

See also:	`Document.implementation`

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`hasFeature()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Implementation.hasFeature() (Method)

A means of enquiring whether a certain feature is supported by the DOM implementation.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myImplementation*.*hasFeature*(*aFeature*, *aVersion*)
Argument list:	*aFeature*	The name of a feature
	aVersion	The feature version

Here is a list of feature and version codes that have been obtained by inspecting the DOM specifications. Note that this list may not be complete or exhaustive as only those values that are mentioned in the DOM specifications published to date have been included:

String	Version	Feature described
CSS	2.0	DOM level 2 CSS support
CSS2	2.0	DOM level support for CSS extended interfaces
Events	2.0	DOM level 2 event model
HTML	1.0	DOM level 1 HTML model
HTML	2.0	DOM level 2 HTML model
HTMLEvents	2.0	DOM level 2 HTML event support
MouseEvents	2.0	DOM level 2 mouse event support (part of Events)
MutationEvents	2.0	DOM level 2 mutation event support (part of Events)
Range	2.0	DOM level 2 text range module
StyleSheets	2.0	DOM level 2 StyleSheets module
Traversal	2.0	DOM level 2 document traversal module
UIEvents	2.0	DOM level 2 user interface event support (part of Events)
Views	2.0	DOM level 2 views module
XML	1.0	DOM level 1 XML extended interfaces
XML	2.0	DOM level 2 XML extended interfaces

See also:	`Event` object, `MouseEvent` object, `MutationEvent` object, `UIEvent` object

Implementation-defined behavior (Definition)

Correct behavior that is specific to a particular implementation.

Some behavior falls outside the ECMAScript standard but is nevertheless correct. Implementers should try to adhere to the spirit of the standard as far as possible. If other standards are built into the implementation (such as DOM for example), the standards working groups generally try to make the standards interoperable, and this helps to make sure that an implementation can support them both easily.

It is important that the implementers properly document any implementation specific behavior.

See also:	Behavior, Host environment, Host object, Implementation-supplied code, Implementation-supplied function

Implementation-supplied code (Definition)

Script source that is provided by the implementation.

Availability:	ECMAScript edition –2

The implementation-supplied code is provided by the hosting environment when it creates an implementation defined function.

As the function is created, the hosting environment may or may not additionally provide a formal parameter list for the function.

On initialization, the scope chain is set up to contain the activation object as its first element.

The caller provides this value, but in some situations the value `null` may be passed. In that case, the global object will be used in its place.

The `ImplicitThis` and `ImplicitParents` attributes affect the way that other lists of objects are attached to the scope chain. They interact to some extent as illustrated in this table:

ImplicitThis	ImplicitParents	First item	Second item	Third item	Fourth item
no	no	`activation` object	`global` object	-	-
no	yes	`activation` object	list of objects provided by this value	`global` object	-
yes	no	`activation` object	this value	`global` object	-
yes	yes	`activation` object	this value	list of objects provided by this value	`global` object

Finally the global object is placed in the scope chain after all other objects.

Variable instantiation is performed using the activation object as the `variable` object and any initial variables are flagged with a `DontDelete` attribute.

See also:	Executable code, Execution context, Implementation-defined behavior

Cross-references:

ECMA 262 edition 2 –section –10.1.1

ECMA 262 edition 2 –section –10.1.2

ECMA 262 edition 2 –section –10.1.6

ECMA 262 edition 2 –section –10.2.4

ECMA 262 edition 3 –section –10.1.1

ECMA 262 edition 3 –section –10.1.2

ECMA 262 edition 3 –section –10.1.6

Implementation-supplied function (Definition)

The script interpreter can provide functions.

Availability:	ECMAScript edition –2

Implementation supplied functions are part of the hosting implementation, although it may depend on the core functionality of the interpreter to provide the necessary services.

The source text for an implementation supplied function is provided by the host environment. The mechanisms by which they are created is host dependent.

The functions created by the implementation may have any combination of `ImplicitThis` and `ImplicitParents` attributes. These control the way that the scope chain is set up when the function is initialized. Note that these are function `object` attributes and `notProperty` attributes. Depending on the host implementation you may or may not have access to these attributes to be able to define their settings.

See also:	`Function` object, `function( ... ) ...`, Implementation-defined behavior, Scope chain

Cross-references:

ECMA 262 edition 2 –section –10.1.1

ECMA 262 edition 3 –section –10.1.1

implements (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Implicit conversion (Definition)

Type conversions that happen automatically.

In JavaScript, the type of a value may be promoted or demoted (coerced) from one kind to another according to the context in which it is used.

A number becomes a string if it is used in a string concatenation.

A string becomes a number if it contains a numeric value and if it is used in an arithmetic expression.

This happens automatically as expressions are evaluated.

The specific behavior depends on the data type of the value at the time the expression is invoked. It also depends on the type of expression being invoked.

This also affects the values yielded by expressions when they are used as operands in other expressions. In this case the value is an ephemeral item not stored in a variable nor in a visible container that the script can access, nevertheless, it behaves as any other value would.

The result of a function call may be affected in the same way when it is used in an expression.

Refer to the descriptions of each of the primitive value types for details of the result of a type conversion on a value of that type.

See also:	Array(), Boolean literal, Boolean(), Cast operator, Date(), Function(), Number(), Numeric literal, Object(), String literal, String(), ToBoolean, ToInt32, ToInteger, ToNumber, ToObject, ToPrimitive, ToString, ToUint16, ToUint32

ImplicitParents (Attribute)

An internal attribute of a function object referred to by an object property.

Availability:	ECMAScript edition –2

This internal attribute controls the way that the scope chain might be modified when an implementation defined function is being executed. If this attribute is set, then the list of objects defined by the `'this'` value will be added to the scope chain after the implementation supplied activation object.

The difference between the behavior of the `ImplicitParents` and the `ImplicitThis` internal attributes is quite subtle and you should consult and fully understand the ECMA standard as it relates to them.

See also:	`ImplicitThis`

Cross-references:

ECMA 262 edition 2 –section –10.1.1

ECMA 262 edition 2 –section –10.2.4

ImplicitThis (Attribute)

An internal `Function` object property.

Availability:	ECMAScript edition –2

This internal attribute controls the way that the scope chain might be modified when an implementation defined function is being executed. If this attribute is set, then the `'this'` value will be added to the scope chain after the implementation supplied activation object.

The difference between the behavior of the `ImplicitParents` and the `ImplicitThis` internal attributes is quite subtle, and you should consult and fully understand the ECMA standard as it relates to them.

See also:	`ImplicitParents`

Cross-references:

ECMA 262 edition 2 –section –10.1.1

ECMA 262 edition 2 –section –10.2.4

import (Statement)

Import some properties that have been exported from another execution context.

Availability:	ECMAScript edition –2 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –3.0

JavaScript syntax:	-	`import anObject.aFunction;`
	-	`import anObject.aProperty;`
Argument list:	*aFunction*	A function object to imprt
	anObject	An object that is exporting some property values
	aProperty	A property value to import

ECMAScript edition 2 suggests this is a future extension. As of the third edition of the ECMAScript standard it is still denoted as a reserved word.

Navigator 4 anticipates that a future standard will endorse this capability and provides it anyway.

A layer might import a function exported by another layer so that they can exchange values or operate on one another.

The imported property name can include the wildcard asterisk character to match several properties.

Warnings:

- This only works in Netscape 4 when the `LANGUAGE` attribute is set to "`JavaScript1.2`". Using import will affect the behavior of the `==` and `!=` operators as well.
- This can affect the security policy regarding the "same-signer" trustworthiness of a page.

See also:	export, Same origin, Signed scripts

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

in (Operator/logical)

Test for the existence of a property in an object.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*aProperty* `in` *anObject*
Argument list:	*aProperty*	A specific property to test for the existence of

The object is examined to see if the property exists. If it does, then a Boolean `true` value is returned, otherwise the expression returns `false`.

This might be useful as a work-around for when you need to test for the existence of a property but to do so by referring to it directly might cause a run-time error.

The logical operator yields `true` if the property exists in the object and `false` if the property is not available.

Warnings:

- This is not available for use server-side with Netscape Enterprise Server 3.

Cross-references:

ECMA 262 edition 3 –section –11.8.7

in ... (Keyword)

Part of the `for ... in` code execution mechanism.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
JavaScript syntax:	-	`for(`*`aVariable`* `in` *`anObject`*`)` `{` *`someCode`* `}`
Argument list:	*`anObject`*	An object to be examined for properties.
	`aVariable`	A variable to store each enumerated property name in
	`someCode`	Some script source to execute for each enumeration

See also:	in, `for( ... in ... ) ...`

Cross-references:

ECMA 262 edition 2 –section –12.6.3

ECMA 262 edition 3 –section –11.8.7

In leap year (Time calculation)

A date and time algorithm defined by ECMAScript.

Availability:	ECMAScript edition –2
Property/method value type:	`Boolean primitive`

It is useful to know whether you are in a leap year. ECMA compliant implementations use an extended Gregorian system to compute dates and can compute leap year flag values internally. This facility may not be available within scripts in all implementations. It can be simulated however.

ECMA compliant implementations use an extended Gregorian system to map a day number to a year number and to determine the month and date within that year. In this system, leap years are summarized thus:

```
DaysInYear(y) =

365 if ((y modulo 4) != 0)

366 if ((y modulo 4) == 0) and ((y modulo 100) != 0)

365 if ((y modulo 4) == 0) and ((y modulo 400) != 0)

366 if ((y modulo 400) == 0)
```

All non-leap-years years have 365 days with the usual number of days in each month. Leap years have an extra day in February. The calculation shown below uses known leap years and non-leap years to adjust the day numbers and yield the day number of the first day of the given year and then use that to work out the time in milliseconds when the year started:

```
DayFromYear(y) =

365 * (y – 1970) +

floor((y – 1969) / 4) -

floor((y – 1901) / 100) +

floor((y – 1601) / 400)

msPerDay = 86400000

TimeFromYear(y) = msPerDay * DayFromYear(y)

YearFromTime(t) = The largest integer y to make TimeFromYear(y) less than or equal
to t.
```

This leap year method would yield a 1 for years that are leap years and 0 for the others:

```
InLeapYear(y) = 0 if DaysInYear(y) == 365

= 1 if DaysInYear(y) == 366
```

To use a time value instead of a year number, the function can be modified like this:

```
InLeapYear(t) = 0 if DaysInYear(YearFromTime(t)) == 365

= 1 if DaysInYear(YearFromTime(t)) == 366
```

Example code:

```
// Test for leap year
document.write("<TABLE BORDER=1>");
for(ii=1980; ii<2009; ii++)
{
   document.write("<TR><TD>");
   document.write(ii);
   document.write("</TD><TD>");
   document.write(inLeapYear(ii));
   document.write("</TD></TR>");
}
document.write("</TABLE>");

// Flag a leap year with a Boolean value
function inLeapYear(aYear)
{
   if((aYear % 4) != 0)
   {
      return false;
   }

   if(((aYear % 100) != 0) ||
      ((aYear % 400) == 0))
   {
      return true;
   }

   return false;
}
```

See also:	Day from year, Days in year, Time range, Year from time, Year number

Cross-references:

ECMA 262 edition 2 –section –15.9.1.3

ECMA 262 edition 3 –section –15.9.1.3

Included JavaScript files (Definition)

The technique of including JavaScript files from external locations.

Refer to:

`<SCRIPT SRC="...">`

Increment value (++) (Operator/postfix)

Pre or post incrementing operator.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`++anOperand`*
	-	*`anOperand`*`++`
Argument list:	*`anOperand`*	A numeric value that can be incremented

The operand is incremented by one.

A prefixing incrementor will add 1 to the operand value before it is used in an expression.

A post-fixing incrementor will add 1 to the operand after it is used in an expression.

Be careful how you use this pre/post placement as you can easily generate 'off by one' errors in your algorithms by placing the operator on the wrong side of the operand.

This operator is more or less functionally equivalent to:

```
anOperand += 1
```

which is equivalent to:

```
anOperand = anOperand + 1
```

See also:	Add then assign (+=), Additive expression, Additive operator, Decrement value (––), Postfix expression, Postfix increment (++), Prefix expression

Cross-references:

ECMA 262 edition 2 –section –11.3.1

ECMA 262 edition 3 –section –11.3.2

Infinity (Constant/static)

A literal constant whose type is a built in primitive value.

Availability:	ECMAScript edition –2 JavaScript –1.3 JScript –3.0 Internet Explorer –4.0 Netscape –4.06	
Property/method value type:	`Number primitive`	
JavaScript syntax:	`-`	`Infinity`

The primitive value Infinity represents the positive infinite number value.

In JavaScript you can use the values positive infinity and negative infinity. They make reference to a global special variable called Infinity, and you can place an optional unary plus or unary minus in front to yield the positive and negative extremes.

If you are in an environment that does not have the `Infinity` value implemented, then you may be able to create one yourself like this:

```
var Infinity = 1e300 * 1e300;
```

You can check for infinity values with `Number.POSITIVE_INFINITY` and `Number.NEGATIVE_INFINITY`. They should be identical to `Infinity` and `-Infinity` which are properties of the `Global` object.

Note that although the type of result when testing the value `Infinity` or the copies available from the `Number` object is number, the value will print as "Inf" when displayed with a `document.write()` method.

Warnings:

- This constant is available as a property of the `global` object in MSIE version 4 but not in Netscape 4.
- This is not available for use server-side with Netscape Enterprise Server 3.
- Note that you can assign a new value to the `Infinity` property on some browsers. This is somewhat dangerous and may cause unpredictable results later on.

See also:	Arithmetic constant, Exception, Global object, Global special variable, isFinite(), NaN, Number, Number, Number.NEGATIVE_INFINITY, Number.POSITIVE_INFINITY, Range error, Special number values, Value property, Zero value

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –4.3.22

ECMA 262 edition 2 –section –15.1.1.2

ECMA 262 edition 3 –section –4.3.22

ECMA 262 edition 3 –section –15.1.1.2

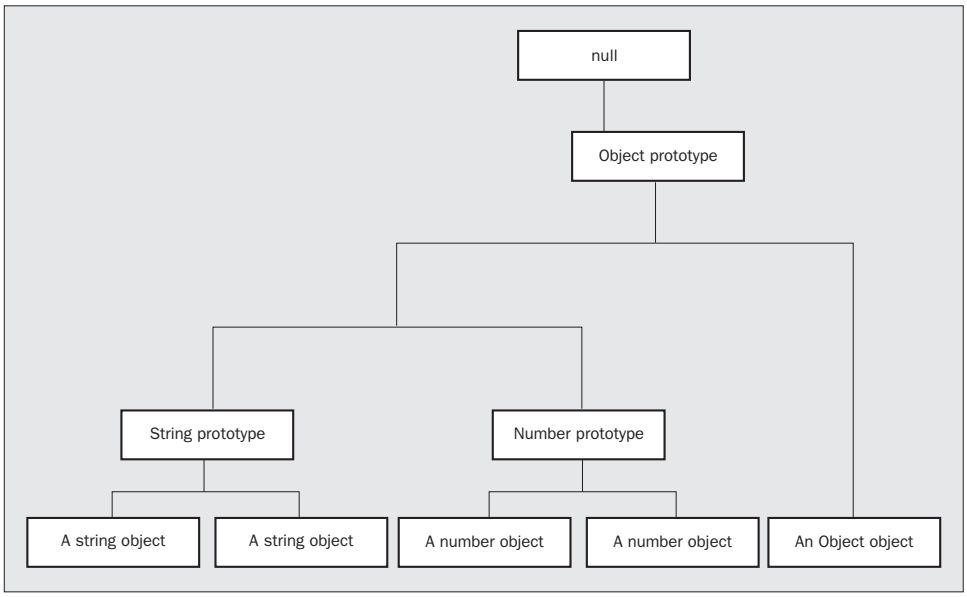
Wrox *Instant JavaScript* –page –14

Inheritance (Definition)

Inheritance is provided through a prototype chain.

Availability: ECMAScript edition –2

Inheritance in JavaScript is implemented by means of a chain of prototypes that link upwards from child to parent until a `null` reference is encountered.



See also: Prototype Based Inheritance

Cross-references:

ECMA 262 edition 2 –section –4.2.1

ECMA 262 edition 3 –section –4.2.1

Initialization (Definition)

Setting a variable to its initial condition.

Initialization code is very similar to assignment statements in the main body of the code. There is a slight difference in that the assignment takes place before any code is executed.

If variables are not initialized they will contain the value undefined.

The initialization happens when the variable is declared using the `var` keyword.

Objects are initialized when they are constructed. This is a somewhat different process to that of initializing a variable.

If a variable is initialized to equal an object, it is really a reference to that object and not the object itself that gets stored in the variable.

See also:	Constant expression, Declaration, `var`

Inline script (Definition)

Code that is embedded in a page or executed during page loading.

Strictly speaking, inline code is code that is embedded into the page in some enclosing `<SCRIPT></SCRIPT>` tags. However, some code can be included from an external file using the `SRC` attribute. This code is not inline code because it no longer lives in the page. However, when the page is loaded, it executed inline as if it had been part of the page all the time.

You may find yourself arguing over the fine points of whether included code is inline code, but to the browser it simply does not matter.

See also:	`<SCRIPT ARCHIVE="...">,<SCRIPT SRC="...">`

Cross-references:

Wrox *Instant JavaScript* –page –43

Inline tags (Definition)

An inline element can be placed anywhere inside a line of text.

An inline tag does not cause a line break in the way that a block level tag does. Inline tags can be placed at the start, end or in the middle of the line. They may cause the line spacing to adjust to accommodate them if they are particularly high.

Here is a list of inline tags:

- `<A>`
- `<EM>`
- `<I>`
- `<IMG>`
- `<SPAN>`
- `<STRONG>`
- `<TT>`

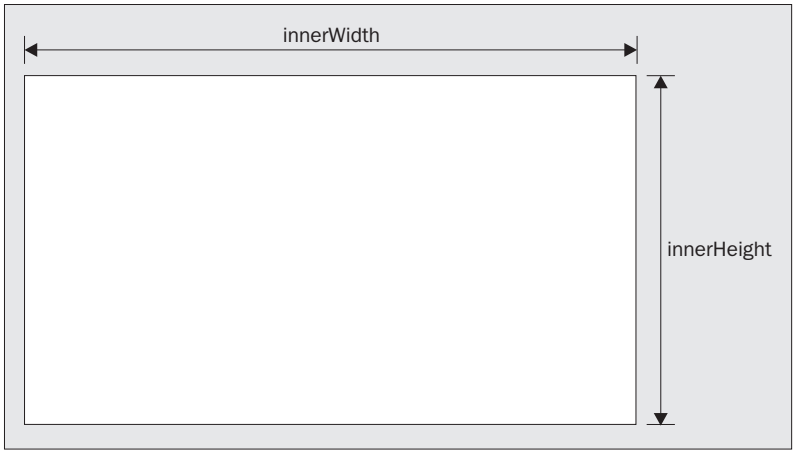
Some block level tags can appear to be inline tags if they control the alignment and text flow around them in such a way as to clear the alignment at one or other side.

Layers complicate things –they can make any block level tag appear to be an inline tag, however, positioning and text flow on separate layers is difficult to control effectively.

innerHeight (Property)

An alias for the `window.innerHeight` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`innerHeight`
	-	*`myWindow`*`.innerHeight`



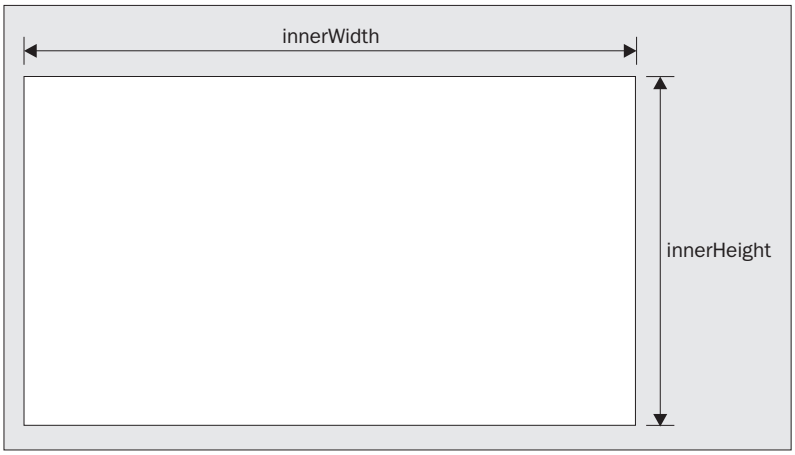
Refer to:

`Window.innerHeight`

innerWidth (Property)

An alias for the `window.innerWidth` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`innerWidth`
	-	*`myWindow`*`.innerWidth`



Refer to:

`Window.innerWidth`

Input event (Definition)

Another name for a raw input event. These are sometimes called HTML control events.

See also:	`onBlur`, `onChange`, `onFocus`, `onReset`, `onSelect`, `onSelectStart`, `onSubmit`, Event propagation

Input object (Object/DOM)

Another name for a `FormElement` object.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0
Inherits from:	`Element` object

JavaScript syntax:	-	*myInput* = *myDocument*.*aFormName*.*anElementName*
	-	*myInput* = *myDocument*.*aFormName*.elements[*anItemIndex*]
	IE	*myInput* = *myDocument*.all.*anElementID*
	IE	*myInput* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*myInput* = *myDocument*.all[*aName*]
	-	*myInput* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*myInput* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*]
	-	*myInput* = *myDocument*.getElementById(*anElementID*)
	-	*myInput* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myInput* = *myInputArray*[*aName*]
	-	*myInput* = *myInputArray*[*anIndex*]
	-	*myInput* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
HTML syntax:	<INPUT TYPE="aType">	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The name attribute of an element
	anElementID	The ID attribute of an element
	anItemIndex	A valid reference to an item in the collection
	aFormIndex	A reference to a particular form in the forms collection
Object properties:	accept, accessKey, align, alt, checked, dataFld, dataFormatAs, dataSrc, defaultChecked, defaultSelected, defaultValue, disabled, form, length, maxLength, name, readOnly, recordNumber, selected, selectedIndex, size, src, status, tabIndex, type, value	
Object methods:	blur(), click(), createTextRange(), focus(), handleEvent(), select()	
Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onChange, onClick, onDblClick, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onRowEnter, onRowExit, onSelect	

This is a generic description of a form element object. The object will really be a concrete manifestation of a particular class, but is available generally as an item in the elements array that belongs to the form.

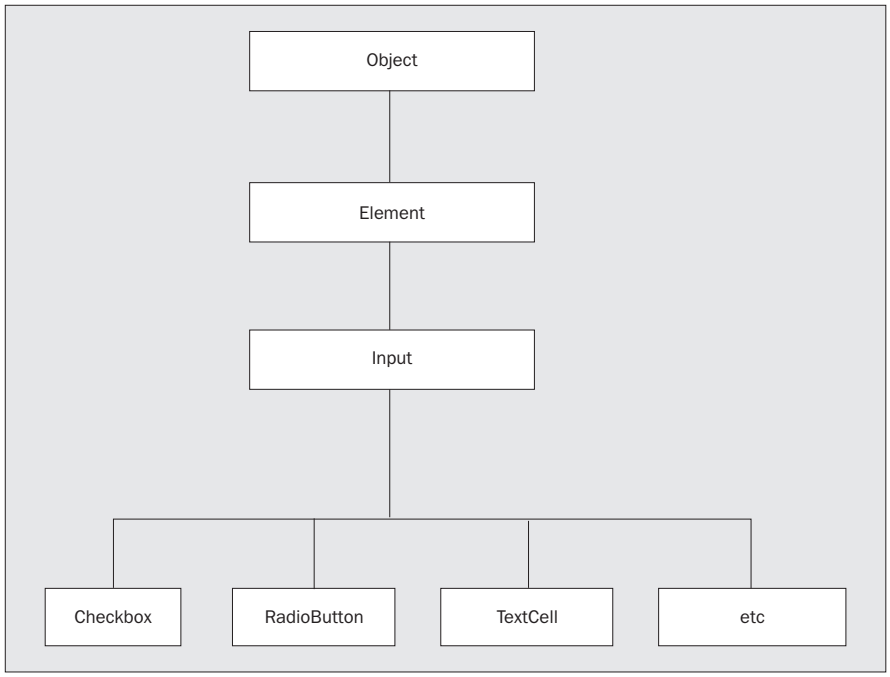
We have tried to conceive a general model of the object relationships in a browser, a difficult task –we document a general purpose class referred to as an `Element` object. Most displayable items in a document that are instantiated by an HTML tag can be considered to be sub-classes of the `Element` object.

Input objects, collectively, are a sub-class of the `Element` object class so to avoid over-duplicating the same coverage, properties, methods, events and collections that are specific to `Input` objects are discussed here and are omitted from the discussion topics relating to the `Element` object. They are listed in the property, method, collection and event summary for the `Element` object.

Likewise, under the `Input` object, those properties, method, collections and events that apply generally to all kinds of `Input` objects are documented here, but those that are specific to only a particular kind of `Input` object sub-class are covered under specific topics relating to that class.

Some properties and methods of the `Input` objects and its specific sub-classes are platform specific. The `dataFld`, `dataSrc` and `dataFormatAs` properties are only available in MSIE. Assigning event handlers to `onevent...` properties may also support different event sets in each browser platform.

MSIE supports an `INPUT` object class rather than an `Input` object class.



Warnings:

- Note that on MSIE, `Input` objects are actually `INPUT` objects because MSIE follows a general rule of naming object classes after the capitalised name of the HTML tag that instantiates them.
- Beware that although a small sub-set of the complete range of properties and methods are supported on all browsers, there are many properties and methods that are only available on one browser or the other.

See also:	`FIELDSET` object, `ISINDEX` object, `Label` object, `Legend` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`accept`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`accessKey`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
alt	1.5 +	3.0 +	6.0 +	3.02 +	-	1 +	-	-
checked	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
dataFld	-	3.0 +	-	4.0 +	-	-	-	-
dataFormatAs	-	3.0 +	-	4.0 +	-	-	-	-
dataSrc	-	3.0 +	-	4.0 +	-	-	-	-
defaultChecked	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
defaultSelected	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	Warning
defaultValue	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	ReadOnly
disabled	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
form	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	ReadOnly
length	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	Warning
maxLength	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
name	1.5 +	5.5 +	6.0 +	5.5 +	3.0 +	1 +	-	-
readOnly	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
recordNumber	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
selected	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	Warning
selectedIndex	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	Warning
size	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	-
src	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
status	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	Warning
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
type	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	1 +	-	ReadOnly
value	1.5 +	1.0 +	6.0 +	3.02 +	3.0 +	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
blur()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
click()	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	1 +	-	-
createTextRange()	-	3.0 +	-	4.0 +	-	-	-	-
focus()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
handleEvent()	1.2 +	-	4.0 +	-	-	-	-	-
select()	1.1 +	3.0 +	3.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	?
onChange	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelect	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-

Inheritance chain:

`Element` object, `Node` object

Input.accept (Property)

Defines an acceptable MIME type to be submitted to the server.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myInput`*`.accept`

Refer to:

`BUTTON.accept`

Input.accessKey (Property)

A key that needs to be pressed before the input object will respond to data entry.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInputObject*`.accessKey`

This is an MSIE extension that allows the input elements to be deadlocked unless a certain key is held down. For some input elements, this key is the one that selects the item, as is the case with the checkbox.

See also:	`!` object, `Anchor` object, `Applet` object, `Area` object, `BODY` object, `Element` object, `Embed` object, `FIELDSET` object, `Label` object, `Legend` object, `MARQUEE` object, `OBJECT` object, `TBODY` object, `TD` object

Input.align (Property)

The alignment of the `paragraph` object with respect to its parent object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInput*`.align`

Input.alt (Property)

An alternative text, used when an image is defined as an input item.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –3.02 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInput*`.alt`

See also:	`BUTTON.alt, IMG.alt`

Input.blur() (Method)

Removes input focus from the input element.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
JavaScript syntax:	-	*myInputObject*`.blur()`

This will trigger the `Blur` event handler function attached to the `onblur` property of the object.

See also:	`Input.focus(),onBlur,onFocus,Window.blur(), Window.focus()`

Input.checked (Property)

The state of the button is returned by this property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myInput*`.checked`

See also:	`Checkbox.checked, RadioButton.checked`

Input.click() (Method)

Sends an artificial mouse click event to the input element.

Availability:	DOM level –1 JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0 Opera –3.0	
JavaScript syntax:	-	*myInputObject*.click()

This will trigger the `Click event` handler function attached to the `onClick` property of the object.

See also:	`onClick`

Input.createTextRange() (Method)

Used in MSIE for creating a text range.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`TextRange object`	
JavaScript syntax:	IE	*myInputObject*.createTextRange()

This method should only be used if the receiving object responds `true` to its `isTextEdit` property request.

See also:	`Element.isTextEdit`, `TextRange` object

Input.dataFld (Property)

This binds the `input` object to a remote data source in MSIE.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myInputObject*.dataFld

This is part of the MSIE data binding mechanism that associates a column name in the data source with the value property of an `Input` object. You must also set the `dataSrc` property for the object. Normally, both the `dataFld` and `dataSrc` values would be defined with the `DATAFLD="..."` and `DATASRC="..."` HTML tag attributes in the document source.

Note that the value is case sensitive and must refer to a column that exists within the data source.

Setting both the `dataFld` and `dataSrc` properties to an empty string will disconnect the element from the database.

Input.dataFormatAs (Property)

Indicates whether data loaded from the database should be treated as text or HTML.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myInputObject*.dataFormatAs

This property can legally only contain one of two values:

- `HTML`
- `text`

Input.dataSrc (Property)

The name of a remote ODBC data source is stored in this property.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myInputObject*.dataSrc

This is part of the MSIE data binding support. It contains the name of an ODBC data source (which might be any kind of SQL database that supports such an adapter). This data source is then associated with the element and various columns of it may provide the values of different element objects by means of their `dataFld` property.

Normally, both the `dataFld` and `dataSrc` values would be defined with the `DATAFLD="..."` and `DATASRC="..."` HTML tag attributes in the document source.

Setting both the `dataFld` and `dataSrc` properties to an empty string will disconnect the element from the database.

Input.defaultChecked (Property)

The default checked state for a radio button or checkbox in a form.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myInput*.defaultChecked

See also:	`Checkbox.defaultChecked`, RadioButton.defaultChecked

Input.defaultValue (Property)

The default value of the `input` object when the page was loaded.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInputObject*.defaultValue
HTML syntax:	`<INPUT VALUE="aValue">`	
Argument list:	*aValue*	A value string for the input element

This is the value of the `Input` element originally furnished by the `VALUE="..."` HTML tag attribute when the page was first loaded. You cannot change this value and it is provided so that a JavaScript can restore the default value or test that the current value is the same or different to it.

Property attributes:

`ReadOnly`.

Input.disabled (Property)

Activates and deactivates the input element within the form.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myInputObject*`.disabled`

If this value is set `true`, then the input element can no longer receive focus or be interacted with. It will also no longer be submitted with the other input elements when the form is submitted back to the server.

Setting this value to `false` renders the input element active again.

See also:	`TextCell.readOnly`

Input.focus() (Method)

Brings input focus back to the input element.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
JavaScript syntax:	-	*myInputObject*`.focus()`

The receiving `Input` element will receive a `Focus` event trigger and execute its function referred to by the onfocus property.

The element that previously had focus (if any element did) will receive a `Blur` event trigger.

See also:	`Input.blur(),onBlur,onFocus,Window.blur(), Window.focus()`

Input.form (Property)

The form object that the `input` element belongs to.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Form object`	
JavaScript syntax:	-	*`myInputObject`*`.form`

Input elements must belong to a containing form. The form is the next outermost `<FORM>` tag (although `<FORM>` tags should not be nested). This is represented by a `Form` object (or a `FORM` object in MSIE).

See also:	`Form` object

Property attributes:

ReadOnly.

Input.handleEvent() (Method)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*`myInputObject`*`.handleEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An event to be handled by this object

On receipt of a call to this method, the input object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`handleEvent()`

Input.maxLength (Property)

The maximum length allowed for an input entry field.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myInput*.maxLength

This defines the maximum number of characters that are allowed to be entered into the input text field. The browsers differ in how they handle this value. Some will warn the user with a beep or flash on the screen, others simply stop accepting keystrokes when this number of characters have been entered.

See also:	`Password.maxLength, TextCell.maxLength`

Input.name (Property)

This corresponds to the `NAME` attribute of the `<INPUT>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInputObject*.name
HTML syntax:	`<INPUT NAME="`*aName*`">`	
Argument list:	*aName*	A name for the input element

Objects are identified either by the `NAME="..."` HTML tag attribute or by the `ID="..."` HTML tag attribute.

Netscape shows a marginal preference for the `name` property while MSIE seems slightly better disposed towards the `ID` property. However, in many cases both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

See also:	`BUTTON.name`

Input.onevent (Property)

Input objects support a variety of different events.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Event handler`	
JavaScript syntax:	-	*myInputObject*`.onevent`

Although this property is named `onevent`, there is actually no such property. All event handlers for `input` objects have names that begin with the 'on' prefix. The name then describes the event.

The properties contain references to a handler function that is invoked for each kind of event. Events that are not handled are represented by a `null` value in the corresponding handler property.

The event model is different for each browser, and the way that events are propagated is different as well. As long as you define only a single handler and keep the event trapping simple, both browsers behave similarly enough that you will be able to realize a portable script implementation.

Note that the event model undergoes a significant revision with the introduction of DOM level 2 capabilities which are currently provided in the new Netscape 6.0 browser.

Refer to the topic describing each kind of `Input` object sub-class for details of the events that this can handle.

Input.readOnly (Property)

Set to `true` if the input text field cannot be changed.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myInput*`.readOnly`

The value of the input text field is defined but cannot be changed by the user.

Do not confuse this with the `ReadOnly` internal attribute that controls whether an object property value can be changed.

See also:	`Password.readOnly`, `ReadOnly`, `TEXTAREA.readOnly`, `TextCell.readOnly`

Input.recordNumber (Property)

The record number within the data set that created the input element's content.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myInputObject`*`.recordNumber`

This is a property that is part of the MSIE data binding support. It contains an integer value that is the record number within the data set that created this object.

This is useful when you are building pages with ASP and Active Data Objects (ADO).

See also:	Active Server Pages, ADO

Property attributes:

`ReadOnly`.

Input.select() (Method)

Triggers a `Select` event on an `input` object.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
JavaScript syntax:	-	*`myInputObject`*`.select()`

If the `Input` object is a `FileUpload`, this selects all the text in the `FileUpload` object so that it can be cut or copied by the user if necessary or used as `TextRange` and have a command executed on it. It also triggers a `Select` event.

See also:	`FileUpload.select(),OptionsArray.select(),`
	`Password.select(),TEXTAREA.select(),`
	`TextCell.select()`

Input.size (Property)

Returns the size of the file to be uploaded when the `Input` object represents a file upload.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInput*`.size`

It may be important to limit the size of files being uploaded to the server.

It is very possible that vindictive users may use this as an opportunity to try and crash your server or deny its availability to other users by uploading massive files to it.

In addition, there may be limits you want to impose as a courtesy to the network managers who run the LANs and WANs where your users will be browsing. Allowing the user to upload unnecessarily large files will saturate their network as well as your server.

By testing this property, you can impose limits of acceptability on file uploads.

See also:	`FileUpload.size`, `Password.size`, `TextCell.size`

Input.src (Property)

The URL where the image is located when an `input` object represents a clickable image.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInput*`.src`

Refer to:

`Image.src`

Input.tabIndex (Property)

A control of where the form input element appears in the tabbing order of the page.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myInputObject*`.tabIndex`

This value indicates where in the tabbing sequence this object and any of its children will be placed. The tabbing order is used when filling in forms. Pressing the `[tab]` key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

Input.type (Property)

The type of a form input element.

Availability:	DOM level –1 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myInputObject*`.type`
HTML syntax:	`<INPUT TYPE="...">`	

The following values can be assigned to the `type` attribute:

- `button`
- `checkbox`
- `file`
- `hidden`
- `image`
- `password`
- `radio`
- `reset`
- `submit`
- `text`
- `textarea`

The `file` value really represents a file-upload.

These values can be used in the `type` property of a `Select` object even though it is not derived from an `Input` object:

- `select-multiple`
- `select-one`

For images, the description of the Input-type in MSIE and Netscape differ because each browser uses a different object class and supports a different set of methods, properties and events.

Refer to the `IMG` object for MSIE and the `Image` object for Netscape.

DOM level 2 specifies that this property should not be read-only. This implies that DOM level 2 allows for the type of an `input` object to change. How the browsers cope with this, given that some of them create objects of specific classes for each type, will be interesting to see.

See also:	`Button.type`, `BUTTON.type`, `Checkbox.type`, `File.Type`, `FileUpload.type`, `Hidden.type`, `Image` object, `Password.type`, `RadioButton.type`, `ResetButton.type`, `selection.type`, `SubmitButton.type`, `TEXTAREA.type`, `TextCell.type`

Property attributes:

`ReadOnly`.

Input.value (Property)

The value associated with the `input` element.

Availability:	DOM level –1 JavaScript –1.5 JScript –1.0 Internet Explorer –3.02 Netscape –6.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myInputObject`*`.value`
HTML syntax:	`<INPUT VALUE="...">`	

When the `input` element is a button, this will be the legend text displayed in the button.

When it is a hidden field, it is the value carried in the hidden field and back to the server again on submitting the form.

See also:	`Button.value`, `BUTTON.value`, `Checkbox.value`, `FileUpload.value`, `Hidden.value`, `Option.value`, `Password.value`, `RadioButton.value`, `ResetButton.value`, `Select.value`, `SubmitButton.value`, `TEXTAREA.value`, `TextCell.value`

Input-output (Definition)

Reading data in and writing data out.

JavaScript was first developed for use in web browsers. Its input and output capabilities in that context are somewhat limited, this mainly being due to the security considerations of allowing a third party to run a script on your computer.

In a web browser implementation, the I/O is limited to being able to access form data, the inner text or HTML of document objects, or perform document writes.

When JavaScript is used server-side, or as a general purpose scripting language for automation, then the implementation adds much more sophisticated I/O capabilities. For example, ScriptEase adds many file and stream based I/O capabilities found in the C language.

See also:	Debugging –client side, Debugging –server side, `Document.write()`, `Document.writeln()`, Error handling

Cross-references:

Wrox *Instant JavaScript* –page –13

InputArray object (Object/browser)

A collection of identically named input elements.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Inherits from:	`Collection` object	
JavaScript syntax:	-	*myInputArray* = *myDocument.aFormName.anElementName*
	IE	*myInputArray* = *myDocument*.all[*aName*]
	-	*myInputArray* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*myInputArray* = *myDocument*.getElementsByName(*aName*)
	-	*myInputArray* = *myDocument*.getElementsByTagName("INPUT")
Argument list:	*aName*	An associative array reference
	aFormIndex	A reference to a particular form in the forms collection
	anIndex	A valid reference to an item in the collection
Object properties:	`length`	

You may encounter this if you have several `Input` objects with the same name. It is better to make sure they have unique names.

Warnings:

- In Netscape, this array is of the type `InputArray`. However, in MSIE it is an object of type `Collection`. Be careful if your code depends on testing object types because you may find it breaks when used across the two browsers.

See also:	`Input` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`length`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	Warning

Inheritance chain:

`Collection` object

INS object (Object/HTML)

An object representing an `<INS>` tag in the document.

Availability:	JScript –3.0 Internet Explorer –4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myINS* = *myDocument*.all.*anElementID*
	IE	*myINS* = *myDocument*.all.tags("INS")[*anIndex*]
	IE	*myINS* = *myDocument*.all[*aName*]
	-	*myINS* = *myDocument*.getElementById(*anElementID*)
	-	*myINS* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myINS* = *myDocument*.getElementsByTagName("INS")[*anIndex*]
HTML syntax:	`<INS> ... </INS>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`cite, dateTime`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

This is a means of marking a section of the page that has been inserted since the previous version of the page. The appearance is styled to indicate the inserted text as distinct from the surrounding text. The `cite` property refers to a document that describes the reason for the insertion.

The DOM level 1 specification includes this in the `ModElement` object functionality.

See also:	`DEL` object, `Element` object, `ModElement` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`cite`	-	5.0 +	-	5.0 +	-	-	-	-
`dateTime`	-	5.0 +	-	5.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.0 +	1.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	1.0 +	2.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

INS.cite (Property)

A string of characters citing the reason for the `<INS>` element being placed into the document.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myINS*`.cite`

The URL of the document that describes why the text being inserted is noted in this property.

INS.dateTime (Property)

The date and time relating to the owning `<INS>` element.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myINS*`.dateTime`

This is the date and time value for when the insertion change occurred. If you are maintaining change control down to the sub-document level in a content management system, these values can be defined from change records in the database.

Inset() (Filter/transition)

A diagonal wipe across the image, revealing a new image.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `Inset()`

Instance method (Definition)

A method belonging to an instance of a class.

Refer to:

Method

Instance variable (Definition)

A variable property belonging to an instance of a class.

Refer to:

Property

instanceof (Operator/logical)

Checks to see if an object is an instance of another object.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *`anObject`* `instanceof` *`anotherObject`*
Argument list:	*`anObject`* — The object to test *`anotherObject the object to test against`* — -

The object referred to by the left operand is examined, and its type compared with the type of the object on the right. The object on the right should be an object with a constructor (that is, a class object) although this likely works with instances as well if they also have constructors inherited.

The ECMA standard (second edition) reserves this keyword for future use in anticipation of the JavaScript 1.4 implementation. That means that a compliant implementation does not need to support this feature.

This operator yields the `true` value if both objects are of the same class, otherwise it is `false`.

Warnings:

- Use with caution. Be aware that there are some bugs in the class instantiation of some objects in Netscape and MSIE. This may always not yield `true` when you expect it to.
- This is not available for use server-side with Netscape Enterprise Server 3.

See also:	Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –11.8.6

Instantiating Function (Definition)

Another name for a constructor mechanism.

Availability:	ECMAScript edition –2

An instantiating function is a `Function` object that can be used as a constructor to make new instances of itself. These new instances are anonymous functions.

See also:	Anonymous function

Cross-references:

ECMA 262 edition 2 –section –10.1.1

ECMA 262 edition 3 –section –10.1.1

int (Reserved word)

Reserved for future language enhancements.

The provision of this reserved keyword suggests that future versions of the ECMAScript standard may provide for a stronger data-typing model than is currently available.

This keyword also represents a Java data type and the `int` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	Identifier, java.lang.Integer, JavaScript language, Lexical element, LiveConnect, Reserved word, Type

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Integer (Definition)

A type of number value.

The inclusion of this reserved keyword in the ECMAScript standard suggests that future versions of ECMAScript may be more strongly typed.

See also:	`double`, `float`, Integer promotion, `long`, `Math.ceil()`, `Math.floor()`, `Number`, Reserved word, `short`

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page –35

Integer arithmetic (Definition)

Arithmetic operations performed on integer values.

Arithmetic in the JavaScript interpreter is basically floating point although many values will be integers. If a numeric value is output then it will automatically be truncated to an integer if there is no fractional part. There are several functions and operators available for explicitly converting floating point values to integer values:

- `Math.abs()` function
- `Math.ceil()` function
- `Math.floor()` function
- `Math.round()` function
- `% Remainder`
- `%= Remainder` and assign to an `LValue`

See also:	Math object, Math.abs(), Math.ceil(), Math.floor(), Math.round(), Mathematics, Remainder (%), Remainder then assign (%=)

Integer constant (Definition)

A numeric value expressed as an integer.

An integer constant is a special case of the arithmetic constant. It's values must be integers –no decimal places.

It could be composed of several components if a constant expression is being created.

Integer constants can be defined as decimal, octal or hexadecimal values. Here are some examples:

- Decimal notation – *`myVariable`* = 150
- Hexadecimal notation – *`myVariable`* = 0xFF
- Octal notation – *`myVariable`* = 0377

With a little creativity in the scripting, you can assign string constants to the value of string objects, then decode the character value to yield an integer constant:

```
myString = new String("A");

myVariable = myString.charCodeAt(0);
```

See also:	Arithmetic constant, Constant, Constant expression, Conversion, Decimal value, Hexadecimal value, Octal value, String.charCodeAt()

Integer promotion (Definition)

The action of converting a value during expression evaluation.

Integer promotion is a concept that is used in compilers where there are many more strongly enforced data typing rules.

All the same, JavaScript does the same thing automatically without you realizing it.

The process of promoting a value is to convert it to a higher resolution data type, so that an expression can be evaluated in that higher data type's context without any loss of value.

For example, adding a floating point and an integer value together would be promoted to a floating point addition. The result would stay as a floating point value. However it is conceivable that adding an integer and two floating points together might yield a result that had a zero value fractional part. That resulting value could then be demoted safely back to an integer with no loss of value.

That would be a value preserving demotion.

Most current implementations of JavaScript do not do a great deal of internal promotion and demotion other than conversions between numeric values and strings. This is because the language is currently a weakly typed language and is somewhat forgiving in the area of data types.

Many reserved words are specified by the ECMA standard. There is a comment in the standard that allows implementations to support these reserved words. This suggests that sometime in the future the standard may allow for more strongly typed data vales. Indeed, some implementations may already provide that functionality and still be ECMAScript compliant.

At such a time that an implementation does more strongly type its data values, then this integer promotion may be more visible.

See also:	Conversion, Integer

Integer-value-remainder (Definition)

Integer rounding functions.

The ECMAScript standard provides several functions and operators that yield an integer value or remainder based on a floating point value:

- `Math.abs()` function
- `Math.ceil()` function
- `Math.floor()` function
- `Math.round()` function
- `% Remainder`
- `%= Remainder` and assign to an `LValue`

See also:	Exponent-log function, Function, `function( ... ) ...`, `Math.abs()`, `Math.ceil()`, `Math.floor()`, `Math.round()`, Mathematics, Power function, Remainder (`%`), Remainder then assign (`%=`), Trigonometric function

interface (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Internal function (Definition)

Internal functions are built into the language interpreter but are private.

Availability:	ECMAScript edition –2

Internal functions are built into the language interpreter.

These may also sometimes be referred to as internal methods.

A host implementation may provide additional non-ECMA compliant built-in functions by extending the `Global` object and other core objects that the ECMA standard defines.

See also:	`Function` object, Internal Method

Cross-references:

ECMA 262 edition 2 –section –10.1.1

ECMA 262 edition 3 –section –10.1.1

Internal Method (Definition)

ECMAScript describes internal methods that are private.

Availability:	ECMAScript edition –2

Internal methods are not exposed to the programmer using ECMA compliant JavaScript. However, some implementations may allow you to view them through a debugging interface. The reserved words suggest that at least some of these internal methods may be made visible at a later stage of the language development.

Run time errors may be generated due to faults in the interpreter logic trying to access internal methods that do not exist.

Property	Parameters	Description
Call	A list of argument values	This method executes some code associated with the object via a function call mechanism. Objects that implement this internal method are called functions.
CanPut	A property name	Returns a Boolean value indicating whether you can put values into the named property.
Class	none	A string value describing what kind of object this is.
Construct	A list of argument values	This constructs an object and is normally invoked by the new operator. Objects that implement this internal method are called constructors.
DefaultValue	A hint value	Returns a default value for the object which should be a primitive value and not an object or a reference to one.
Delete	A property name	Removes the specified property from the object.
Get	A property name	This method returns the value of the named property.
HasInstance	A value	Returns a Boolean value indicating whether there is an instance. This only applies to `Function` objects.
HasProperty	A property name	Returns a Boolean value indicating whether that property is available as a member of this object.
Match	A string and an index	Tests for a regular expression and returns a match result.
Prototype	none	Returns the prototype of this object.
Put	A property name and a value	This method stores the value in the named property.
Scope	none	A scope chain for the `Function` object to be executed in.
Value	none	Internal state information associated with this object.

The table describes the base set of ECMA compliant internal methods. Hosted implementations may add to these and implement special methods in any way they need to, however they are not required to implement all of these methods.

Every object must implement the `Get`, `Put`, `HasProperty`, `Delete` and `DefaultValue` functions. This also applies to host objects. Note that the `DefaultValue` method may for some objects generate a runtime error.

ECMA edition 3 introduces the following internal methods/properties:

- `HasInstance`
- `Scope`
- `Match`

See also:	`Call`, `CanPut()`, Construct, `DefaultValue()`, `Delete()`, `Get()`, `HasProperty()`, Internal function, Internal Property, Object, `Put()`

Cross-references:

ECMA 262 edition 2 –section –8.6.2

ECMA 262 edition 3 –section –8.6.2

Internal Property (Definition)

ECMAScript describes internal methods that are private.

Availability:	ECMAScript edition –2

Internal properties are not exposed to the programmer using ECMA compliant JavaScript. However, some implementations may allow you to view them through a debugging interface. The reserved words suggest that at least some of these internal properties may be made visible at a later stage of the language development.

Run time errors may be generated due to faults in the interpreter logic trying to access internal methods that do not exist.

Property	Parameters	Description
Call	A list of argument values	This method executes some code associated with the object via a function call mechanism. Objects that implement this internal method are called functions.
CanPut	A property name	Returns a Boolean value indicating whether you can put values into the named property.
Class	none	A string value describing what kind of object this is.
Construct	A list of argument values	This constructs an object and is normally invoked by the new operator. Objects that implement this internal method are called constructors.
DefaultValue	A hint value	Returns a default value for the object which should be a primitive value and not an object or a reference to one.
Delete	A property name	Removes the specified property from the object.
Get	A property name	This method returns the value of the named property.
HasInstance	A value	Returns a Boolean value indicating whether there is an instance. This only applies to `Function` objects.
HasProperty	A property name	Returns a Boolean value indicating whether that property is available as a member of this object.
Match	A string and an index	Tests for a regular expression and returns a match result.
Prototype	none	Returns the prototype of this object.
Put	A property name and a value	This method stores the value in the named property.
Scope	none	A scope chain for the `Function` object to be executed in.
Value	none	Internal state information associated with this object.

The table describes the base set of ECMA compliant internal properties. Hosted implementations may add to these and implement special properties in any way they need to. Host implementations may not implement all of these properties.

Every object must implement the `Class` property.

The value of the `Prototype` property must be either an object or `null`. Every prototype chain must have finite length, that is to say starting from any object, traversing the prototype chain must ultimately yield a `null` value. Whether or not a native object can refer to a host object as its prototype is implementation dependent.

ECMA edition 3 introduces the following internal methods/properties:

- `HasInstance`
- `Scope`
- `Match`

See also:	Class, class, Internal Method, Object, `prototype` property, `Value` property

Cross-references:

ECMA 262 edition 2 –section –8.6.2

ECMA 262 edition 3 –section –8.6.2

Internet Explorer (Web browser)

A Microsoft web browser product.

In this publication, we refer to this browser by the commonly used abbreviation MSIE.

Many values that MSIE exposes as JavaScript properties reflect the value of an HTML tag attribute. Likewise, many of its special objects are counterparts to the HTML tags.

Where the information is available, the version number that objects, properties and methods became accessible to JavaScript is indicated. In many cases, this may be a later version than when the instantiating HTML tag or attribute was first supported by the browser.

Because the Windows platform in particular and Microsoft products in general are componentized, the JScript interpreter can be replaced over the top of the Internet Explorer browser. This means that you can be running a version of JScript that is later than the browser version you are using. Scripts will work although they may not be able to exploit features of the later browser. For example, JScript 5.5 can be installed over the top of JScript 5.0 in a version 5.0 MSIE browser.

We wrote many scripts to inspect and enumerate various properties of the objects in the MSIE and Netscape browsers –the exposed object types and properties that were hitherto undocumented. They may have been available in earlier versions of the browser, however, where language elements were discovered for the first time, they are initially documented as being available from version 5 of MSIE. A limited amount of further testing was applied where it was suspected that language elements may have been available in earlier releases and the availability modified accordingly.

Because the MSIE browser is componentized to the extent that the JSscript interpreter is actually a separate installation to the MSIE browser, it is very difficult to arrive at a definitive version of JScript that correlates with a particular version of MSIE. For example, several different versions of JScript were extant with the version 3 MSIE browser.

Perhaps this may become less important as browsers converge on a single standard benchmark of functionality. For the time being, current practice suggests that version 4 browsers are rapidly being taken over by version 5 MSIE browsers. Version 2 and 3 of MSIE have declined to such small usage levels as to not require any further serious attempts to support them on new projects.

Refer to the JScript version topic for details of interpreter vs browser revisions.

Warnings:

- Version 3.02 has these problems with identifier names:
  - Do not use dollar signs in identifier names.
  - Identifier names are caseless and this version cannot tell the difference between `AAA` and `aaa`.

See also:	Browser version compatibility, Identifier, JScript version, Platform, Script execution, Web browser

Cross-references:

Wrox *Instant JavaScript* –page –14

Internet Information Server (Product)

Internet Information Server. A Microsoft server product.

Version 4 of IIS is reputed to be 100% ECMAScript compliant.

See also:	Server-side JavaScript

Cross-references:

Wrox *Instant JavaScript* –page –64

Interpret (Definition)

The act of parsing a JavaScript script source text.

To execute a JavaScript script source text, you first have to convert its textual representation into a series of executable steps. This interpretation phase involves the stripping out of commented blocks and the division of the script into tokens –each token is then evaluated and executed sequentially.

Interpretation is distinct from compiling a program. A compiler renders the interpreted code down to a machine-readable form that can be executed directly in terms of CPU opcodes. An interpreted program partially compiles the program to an intermediate form and may store this interpreted data as a series of byte codes. These are then executed in a virtual machine.

Java applets work like this and so do many JavaScript implementations although you cannot normally store the byte-coded version of a JavaScript script. This is beginning to change however and some interactive TV platforms are delivering byte-code forms of JavaScript sourced software to set-top boxes. This is particularly appropriate, since the available bandwidth is much reduced compared to web delivery and a byte-code form takes less time to transmit.

The same applies to the delivery of WMLScript code to WAP enabled mobile phones.

See also:	ATVEF, DVB-MHP, Implementation, Liberate TV Navigator, OpenTV, WAP, WebTV, WML, WMLScript

Interval handlers (Definition)

You can set event handlers to be called periodically on a regular basis.

This is a mechanism for scheduling the execution of a script at regular intervals. The interval is specified with the `setInterval()` method.

See also:	`Window.clearInterval(),Window.setInterval()`

Intrinsic events (Definition)

Those events that pertain to HTML rather than any other host-based triggers.

See also:	Event, Event model

Invert() (Filter/visual)

A visual filter for inverting image colors.

Availability:	JScript –3.0 Internet Explorer –4.0

Refer to:

Filter – `Invert()`

Invoke a function (Definition)

To execute a function when the script is running.

Refer to:

Function call

Iris() (Filter/transition)

A transition effect with the appearance of an iris opening or closing.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `Iris()`

isAlnum() (Simulated functionality)

Check if a character is a number or letter.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However it is useful to script developers as well, and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is a letter or a number. A letter is any character for which the `isAlpha()` function returns `true`. Likewise a number is any character for which `isDigit()` returns `true`.

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for letters and digits
function isAlnum(aChar)
{
   return (isDigit(aChar) || isAlpha(aChar));
}

// Test for digits
function isDigit(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 47) && (myCharCode <  58))
   {
      return true;
   }

   return false;
}
```

```
// Test for letters (only good up to char 127)
function isAlpha(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if(((myCharCode > 64) && (myCharCode <  91)) ||
      ((myCharCode > 96) && (myCharCode < 123)))
   {
     return true;
   }

   return false;
}

alert(isAlnum("5"));
alert(isAlnum("a"));
alert(isAlnum("%"));
```

See also:	Character handling, Character testing, Enquiry functions, `isAlpha()`, `isDigit()`, `isPunct()`, Letter

isAlpha() (Simulated functionality)

Check if a character is a letter.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is a letter. It will return `true` for any character in the following set:

`A B C D E F G H I J K L M`

`N O P Q R S T U V W X Y Z`

`a b c d e f g h i j k l m`

`n o p q r s t u v w x y z`

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for letters (only good up to char 127)
function isAlpha(aChar)
{
   myCharCode = aChar.charCodeAt(0);
```

```
    if(((myCharCode > 64) && (myCharCode <  91)) ||
       ((myCharCode > 96) && (myCharCode < 123)))
    {
       return true;
    }

    return false;
}

alert(isAlpha("a"));
alert(isAlpha("5"));
```

See also:	Character handling, Character testing, Enquiry functions, `isAlnum()`, Letter

isCtrl() (Simulated functionality)

Check if a character is a control code.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is a control code. It will return `true` for any character in the following set:

Value	Character	Meaning
01	<ctrl-A>	SOH
03	<ctrl-C>	ETX
04	<ctrl-D>	EOT
05	<ctrl-E>	ENQ
06	<ctrl-F>	ACK
07	<ctrl-G>	BEL
08	<ctrl-H>	BS
09	<ctrl-I>	HT
0A	<ctrl-J>	LF
0B	<ctrl-K>	VT
0C	<ctrl-L>	FF
0D	<ctrl-M>	CR
0E	<ctrl-N>	SO
0F	<ctrl-O>	SI
10	<ctrl-P>	DLE
11	<ctrl-Q>	DC1

Table continued on following page

Value	Character	Meaning
12	<ctrl-R>	DC2
13	<ctrl-S>	DC3
14	<ctrl-T>	DC4
15	<ctrl-U>	NAK
16	<ctrl-V>	SYN
17	<ctrl-W>	ETB
18	<ctrl-X>	CAN
19	<ctrl-Y>	EM
1A	<ctrl-Z>	SUB
1B	<ctrl-[>	ESC
1C	<ctrl-\>	FS
1D	<ctrl-]>	GS
1E	<ctrl-^>	RS
1F	<ctrl-_>	US
7F	-	DEL

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for control codes
function isCtrl(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if(((myCharCode > -1) && (myCharCode <  32)) ||
      ((myCharCode == 127)))
   {
      return true;
   }

   return false;
}

alert(isCtrl("\u0011"));
alert(isCtrl("a"));
```

See also:	Character handling, Character testing, Control character, Enquiry functions, Letter

isDigit() (Simulated functionality)

Check if a character is a digit.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is a digit. It will return `true` for any character in the following set:

```
0 1 2 3 4 5 6 7 8 9
```

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for digits
function isDigit(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 47) && (myCharCode <  58))
   {
      return true;
   }

   return false;
}

alert(isDigit("a"));
alert(isDigit("3"));
```

See also:	Character handling, Character testing, Enquiry functions, `isAlnum()`, `isODigit()`, `isXDigit()`, Letter

isElementProperty() (Simulated functionality)

A function that tells you whether a property name is inherited from `Element` objects.

Property/method value type:	Boolean primitive

This function switches according to the value of a text string that is passed in. If the string is in the set of strings that enumerate properties that belong to an `Element` object then the value `true` will be returned. This indicates that the property is defined in a super-class.

You can build property enumeration loops that test for this value to build debugging displays of object properties that are less cluttered than simply displaying all the object properties.

The result of this method is `true` if the property has been inherited from the `Element` object otherwise the `false` value is returned.

Example code:

```
function isElementProperty(aProperty)
{
   switch(aProperty)
   {
      case "all"             :
      case "attributes"      :
      case "childNodes"      :
      case "children"        :
      case "className"       :
      case "currentStyle"    :
      case "dir"             :
      case "document"        :
      case "filters"         :
      case "firstChild"      :
      case "id"              :
      case "innerHTML"       :
      case "innerText"       :
      case "isTextEdit"      :
      case "lang"            :
      case "language"        :
      case "lastChild"       :
      case "nextSibling"     :
      case "nodeName"        :
      case "nodeType"        :
      case "nodeValue"       :
      case "offsetHeight"    :
      case "offsetLeft"      :
      case "offsetParent"    :
      case "offsetTop"       :
      case "offsetWidth"     :
      case "outerHTML"       :
      case "outerText"       :
      case "ownerDocument"   :
      case "parentNode"      :
      case "parentElement"   :
      case "parentTextEdit"  :
      case "previousSibling" :
      case "sourceIndex"     :
      case "style"           :
      case "tagName"         :
      case "title"           :
         return true;
   }
   return false;
}
```

See also:	`Element` object

isFinite() (Function/global)

Test a numeric value for infinity.

Availability:	ECMAScript edition –2 JavaScript –1.3 JScript –3.0 Internet Explorer –4.0 Netscape –4.06	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`isFinite(`*`aValueToTest`*`)`
Argument list:	*`aValueToTest`*	A numeric value to check for validity

This is a built-in function to check for the infinity value. Because it is a member of the `Global` object, and the `Global` object is permanently in the prototype inheritance chain, you don't need to identify which object the function belongs to.

Applies the internal `ToNumber` operator to its argument, then returns `true` or `false` depending on whether the value is a finite number or not.

The result is `true` for a valid and finite numeric value and false if the value is `NaN` or one of the Infinities.

Warnings:

- This is not available for use server-side with Netscape Enterprise Server 3.

See also:	Enquiry functions, Function property, `Global` object, Infinity, Special type, `ToNumber`

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.1.2.7

ECMA 262 edition 3 –section –15.1.2.5

isGraph() (Simulated functionality)

Check if a character is a printable glyph.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is printable. However it does exclude the space character from the set of valid characters that return `true`. It is likely that implementation provided versions of this will only support the lower 255 characters in the Unicode character set. Some may only support the lower 128 that correspond to the ASCII character set.

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for visibly printable characters
// (only good up to char 127)
function isGraph(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 32) && (myCharCode <  127))
   {
      return true;
   }

return false;
}
```

See also:	Character handling, Character testing, Control character, Enquiry functions, `isPrint()`, `isPunct()`, Letter

ISINDEX object (Object/HTML)

A deprecated object that represents the `<ISINDEX>` tag. Do not use this in new projects. This tag describes text entry field with an associated prompting text.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0 Deprecated
Inherits from:	`Element` object

JavaScript syntax:	IE	*myISINDEX* = *myDocument*.all.*anElementID*
	IE	*myISINDEX* = *myDocument*.all.tags("ISINDEX")[*anIndex*]
	IE	*myISINDEX* = *myDocument*.all[*aName*]
	-	*myISINDEX* = *myDocument*.getElementById(*anElementID*)
	-	*myISINDEX* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myISINDEX* = *myDocument*.getElementsByTagName("ISINDEX")[*anIndex*]
HTML syntax:	<ISINDEX>	
Object properties:	prompt, form, prompt	
Event handlers:	onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp	

Warnings:

- This element object is deprecated as of HTML version 4.0 and should not be used for any new projects. Refer to the <INPUT> tag and its corresponding objects for a better and more functional replacement.

See also:	Input object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
prompt	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Deprecated
form	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly, Deprecated
prompt	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Deprecated

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

ISINDEX.form (Property)

Returns a reference to a containing `Form` object. If there is none, then a `null` is returned instead.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0 Deprrecated	
Property/method value type:	`Form` object	
JavaScript syntax:	-	*myISINDEX*.form

Property attributes:

`ReadOnly`.

Refer to:

`Form` object

ISINDEX.prompt (Property)

A property relating to the deprecated `<ISINDEX>` object. Avoid the use of this object and property in new projects. This property provides a prompting text message for the text entry field.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myISINDEX*.prompt

Refer to:

`Input` object

isInNet() (Function/proxy.pac)

This is a convenience function for use with `proxy.pac` files.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	`isInNet(`*`aHostname`*`, `*`aSubNet`*`)`
Argument list:	*`aHostname`*	A host name whose IP address is compared with the sub-net
	`aSubNet`	A sub-net to test against the host IP address

This function performs a name to IP translation by means of an IP lookup. This involves connecting to a name server and waiting for its reply. This is not something you will want to do often as it can severely affect your performance.

Once the host IP address is known, it can be checked against the sub-net value and a `true` or `false` value returned.

The value `true` is returned if the host is a valid member of the sub-net and `false` if it is part of another sub-net.

See also:	`FindProxyForURL()`, `isPlainHostName()`, Proxies, `proxy.pac`

Cross-references:

Wrox *Instant JavaScript* –page –58

isLower() (Simulated functionality)

Check if character is lower case.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However, it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function will return `true` for any character in the following set:

```
a b c d e f g h i j k l m

n o p q r s t u v w x y z
```

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for lowercase letters (only good up to char 127)
function isLower(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 96) && (myCharCode < 123))
   {
      return true;
   }

   return false;
}
```

See also:	ASCII, Character handling, Character set, Character testing, Character-case mapping, Enquiry functions, isUpper(), Letter, Unicode

isNaN() (Function/global)

Test a numeric value for validity.

Availability:	ECMAScript edition –1 JavaScript –1.3 JScript –1.0 Internet Explorer –3.02 Netscape –2 Netscape Enterprise Server version –2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`isNaN(`*`aValueToTest`*`)`
Argument list:	*`aValueToTest`*	A numeric value to be tested for validity

This is a built-in function to check for the Not-a-Number value. Because it is a member of the `Global` object, and the `Global` object is permanently in the prototype inheritance chain, you don't need to identify which object the function belongs to.

Applies the internal `ToNumber` operator to its argument and returns `true` or `false` depending on whether the value is a number or not.

The values may not always yield the result you expect.

These all yield a `false` value:

- "4"
- true
- false

- 100.00
- Infinity
- null

These all yield a `true` value:

- "4A"
- "true"
- "false"
- undefined

The result is `true` for valid numeric values and `false` for invalid numerics.

Warnings:

- This is not very useful on MSIE version 3.02 since it cannot understand what a `NaN` is in the first place.
- Although this was added to some browsers in version 1.0 of JavaScript, it was not available in all versions of Netscape until it was added to the Unix variants for JavaScript 1.1. Most browsers being used these days are at least version 1.1 compliant so this issue is becoming less important.
- Comparing the `NaN` value with anything using the == operator will always yield the Boolean `false` value.

See also:	Enquiry functions, Function property, `Global` object, `NaN`, Number, Special type, `ToNumber`

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.1.2.6

ECMA 262 edition 3 –section –15.1.2.4

Wrox *Instant JavaScript* –page –15

Wrox *Instant JavaScript* –page –28

ISO 3166 (Standard)

An ISO standard that identifies different countries.

The ISO 3166 standard is revised occasionally to add new country codes. The following table summarizes the third edition of the standard. Note that some countries will have changed since this edition:

Code	Country
AD	Andorra
AE	United Arab Emirates
AF	Afghanistan
AG	Antingua and Barbuda
AI	Anguilla
AL	Albania
AN	Netherlands Antilles
AO	Angola
AQ	Antarctica
AR	Argentina
AS	American Samoa
AT	Austria
AU	Australia
AW	Aruba
BB	Barbados
BD	Bangladesh
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BH	Bahrain
BI	Burundi
BJ	Benin
BN	Brunei Darussalam
BO	Bolivia
BR	Brazil
BS	Bahamas
BT	Bhutan
BU	Burma
BV	Bouvet Island
BW	Botswana
BY	Byelorussian Ssr
BZ	Belize
CA	Canada
CC	Cocos (Keeling) Islands
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Cote D'ivoire

Table continued on following page

Code	Country
CK	Cook Islands
CL	Chile
CM	Cameroon
CN	China
CO	Colombia
CR	Costa Rica
CS	Czechoslovakia
CU	Cuba
CV	Cape Verde
CX	Christmas Island
CY	Cyprus
DD	German Democratic Republic
DE	Federal Republic Of Germany
DJ	Djibouti
DK	Denmark
DM	Dominica
DO	Dominican Republic
DZ	Algeria
EC	Ecuador
EG	Egypt
EH	Western Sahara
ES	Spain
ET	Ethiopia
FI	Finland
FJ	Fiji
FK	Falkland Islands
FM	Micronesia
FO	Faroe Islands
FR	France
GA	Gabon
GB	United Kingdom of Great Britan and Northern Ireland
GD	Grenada
GF	French Guiana
GH	Ghana
GI	Gibraltar
GL	Greenland
GM	Gambia
GN	Guinea

Table continued on following page

Code	Country
GP	Guadaloupe
GQ	Equatorial Guinea
GR	Greece
GT	Guatemala
GU	Guam
GW	Guinea-Bissau
GY	Guyana
HK	Hong Kong
HM	Heard and Mc Donald Islands
HN	Honduras
HT	Haiti
HU	Hungary
ID	Indonesia
IE	Ireland
IL	Israel
IN	India
IO	British Indian Ocean Territory
IQ	Iraq
IR	Islamic Republic Of Iran
IS	Iceland
IT	Italy
JM	Jamaica
JO	Jordan
JP	Japan
KE	Kenya
KH	Democratic Kampuchea
KI	Kiribati
KM	Comoros
KN	St Kitts and Nevis
KP	Democratic People's Republic Of Korea
KR	Republic Of Korea
KW	Kuwait
KY	Cayman Islands
LA	Lao People's Democratic Republic
LB	Lebanon
LC	Saint Lucia
LI	Lichtenstein
LK	Sri Lanka

Table continued on following page

Code	Country
LR	Liberia
LS	Lesotho
LU	Luxembourg
LY	Libyan Arab Jamahiriya
MA	Morocco
MC	Monaco
MG	Madagascar
MH	Marshall Islands
ML	Mali
MN	Mongolia
MO	Macau
MP	Northern Mariana Islands
MQ	Martinique
MR	Mauritania
MS	Montserrat
MT	Malta
MU	Mauritius
MV	Maldives
MW	Malawi
MX	Mexico
MY	Malaysia
MZ	Mozambique
NA	Namibia
NC	New Caledonia
NE	Niger
NF	Norfolk Island
NG	Nigeria
NI	Nicaragua
NL	Netherlands
NO	Norway
NP	Nepal
NR	Nauru
NU	Niue
NZ	New Zealand
OM	Oman
PA	Panama
PE	Peru
PF	French Polynesia

Table continued on following page

Code	Country
PG	Papua New Guinea
PH	Philippines
PK	Pakistan
PL	Poland
PM	St Pierre and Miquelon
PN	Pitcairn
PR	Puerto Rico
PT	Portugal
PW	Palau
PY	Paraguay
QA	Qatar
RE	Reunion
RO	Romania
RW	Rwanda
SA	Saudi Arabia
SB	Solomon Islands
SC	Seychelles
SD	Sudan
SE	Sweden
SG	Singapore
SH	St Helena
SJ	Svalbard and Jan Mayen Islands
SL	Sierra Leone
SM	San Marino
SN	Senegal
SO	Somalia
SR	Suriname
ST	Sao Tome and Principe
SU	USSR replaced by individual country codes
SV	El Salvador
SY	Syrian Arab Republic
SZ	Swaziland
TC	Turks and Caicos Islands
TD	Chad
TF	French Southern Territories
TG	Togo
TH	Thailand
TK	Tokelau

Table continued on following page

Code	Country
TN	Tunisia
TO	Tonga
TP	East Timor
TR	Turkey
TT	Trinidad and Tobago
TV	Tuvalu
TW	Taiwan
TZ	United Republic Of Tanzania
UA	Ukraine SSR
UG	Uganda
UM	United States Minor Outlying Islands
US	United States of America
UY	Uruguay
VA	Vatican City State
VC	St Vincent and The Grenadines
VE	Venezuela
VG	British Virgin Islands
VI	Virgin Islands (US)
VN	Vietnam
VU	Vanuatu
WF	Wallis and Futuna Islands
WS	Samoa
YD	Democratic Yemen
YE	Yemen
YU	Yugoslavia
ZA	South Africa
ZM	Zambia
ZR	Zaire
ZW	Zimbabwe

See also:	Language codes, `Navigator.systemLanguage`, `Navigator.userLanguage`

ISO 639 (Standard)

An ISO standard that defines language code values.

See also:	ISO 3166, Language codes

isObjectEqual() (Simulated functionality)

Compare the properties of two objects for equality.

Property/method value type:	Boolean primitive

This function tests two objects for equality. This is not the same as identity. Two objects may be deep copies of one another but still be distinctly different objects. If they have the same properties because they are both of the same class, you can compare their enumerable properties for equality.

The example code is fairly simple and could be enhanced in several ways. For example, you should possibly enumerate the second object as well if the first pass proves to yield a `true` value. This would ensure that if the second object has additional attributes, the test will properly yield a Boolean `false` result.

You could also test for object type and return `false`. This would allow the test to be applied to dissimilar objects and still yield a meaningful result.

The result will be `true` if all enumerable properties are equal and `false` if any of them are not.

Example code:

```
// Test objects for equality
function isObjectEqual(anObject1, anObject2)
{
   for(myProp in anObject1)
   {
      if(anObject1[myProp] != anObject2[myProp])
      {
         return false;
      }
   }
   return true;
}
```

See also:	Element.currentStyle

isODigit() (Simulated functionality)

Check if character is an octal numeral.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is an octal digit. It will return `true` for any character in the following set:

```
0 1 2 3 4 5 6 7
```

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for octal only digits
function isODigit(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 47) && (myCharCode <  56))
   {
      return true;
   }

   return false;
}
```

See also:	Character handling, Character testing, Enquiry functions, `isDigit()`, `isXDigit()`, Letter

isPlainHostName() (Function/proxy.pac)

This is a convenience function for use with `proxy.pac` files.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	`isPlainHostName(`*`aHostname`*`)`
Argument list:	*`aHostname`*	The name of a host to check

This function will check to see if the passed-in `HostName` parameter is just a host name on its own or if it has a full domain specified too. This functionality can probably be simulated outside of a `proxy.pac` file by testing for the existence of a full stop within the passed `HostName` parameter.

This function returns `true` if the `HostName` parameter contains a plain host name with no domain specified. If the `HostName` parameter specifies a domain as well then this method returns `false`.

See also:	`FindProxyForURL()`, `isInNet()`, `proxy.pac`

Cross-references:

Wrox *Instant JavaScript* –page –58

isPrint() (Simulated functionality)

Check if a character is printable.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However, it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is printable. Unlike the `isgraph()` function, however it does include the space character in the set of valid characters that return `true`. It is likely that implementation provided versions of this will only support the lower 255 characters in the Unicode character set. Some may only support the lower 128 that correspond to the ASCII character set.

Functionally then, it is the result of testing for space and then cascading into the `isgraph()` test.

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for printable characters (only good up to char 127)
function isPrint(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 31) && (myCharCode <  127))
   {
      return true;
   }

   return false;
}
```

See also:	Character handling, Character testing, Control character, Enquiry functions, isGraph(), Letter, Printing character

isPunct() (Simulated functionality)

Check if a character is a punctuation mark.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However, it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

A punctuation character is any character in the `isGraph()` set except for those in the `isAlnum()` set.

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for punctuation characters
function isPunct(aChar)
{
   return (isGraph(aChar) && !(isAlnum(aChar)));
}

// Test for printable characters (only good up to char 127)
function isGraph(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 32) && (myCharCode <  127))
   {
      return true;
   }

   return false;
}

// Test for letters and digits
function isAlnum(aChar)
{
   return (isDigit(aChar) || isAlpha(aChar));
}

// Test for digits
function isDigit(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 47) && (myCharCode <  58))
   {
      return true;
   }

   return false;
}

// Test for letters (only good up to char 127)
function isAlpha(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if(((myCharCode > 64) && (myCharCode <  91)) ||
      ((myCharCode > 96) && (myCharCode < 123)))
   {
      return true;
   }
   return false;
}
```

See also:	Character handling, Character testing, Enquiry functions, isAlnum(), isGraph(), Letter

isSpace() (Simulated functionality)

Check if a character is whitespace.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However, it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

Any whitespace character should return `true` for this function. The following character codes are considered to be whitespace:

- The space character
- A form feed character
- A Newline character
- A carriage return
- A tab character
- A vertical tab

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for space characters
function isSpace(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if(((myCharCode >  8) && (myCharCode < 14)) ||
      (myCharCode == 32))
   {
      return true;
   }

return false;
}
```

See also:	Character handling, Character testing, Enquiry functions, Letter

isUpper() (Simulated functionality)

Check if a character is upper case.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However, it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function will return `true` for any character in the following set:

```
A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z
```

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for upper case letters (only good up to char 127)
function isUpper(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 64) && (myCharCode <  91))
   {
      return true;
   }

   return false;
}
```

See also:	ASCII, Character handling, Character set, Character testing, Character-case mapping, Enquiry functions, `isLower()`, Letter, Unicode

isXDigit() (Simulated functionality)

Check if character is a hexadecimal numeral.

Property/method value type:	Boolean primitive

This is a function normally found in the C language. However, it is useful to script developers as well and may be available in some implementations as an extension to the ECMA standard functionality.

This function tests to see if a character is a hexadecimal digit. It will return `true` for any character in the following set:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Strictly speaking, this function should be coded to be aware of locale specific issues. You may want to take the example simulation provided here and modify it to your own needs to support that. This is just a basic working example.

Example code:

```
// Test for hexadecimal digits
function isXDigit(aChar)
{
   return (isDigit(aChar) || isA2F(aChar));
}
```

```
// Test for digits
function isDigit(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if((myCharCode > 47) && (myCharCode <  58))
   {
      return true;
   }

   return false;
}

// Test for letters A to F
function isA2F(aChar)
{
   myCharCode = aChar.charCodeAt(0);

   if(((myCharCode > 64) && (myCharCode <  71)) ||
      ((myCharCode > 96) && (myCharCode < 103)))
   {
      return true;
   }

   return false;
}

alert(isXDigit("G"));
alert(isXDigit("A"));
```

See also:	Character handling, Character testing, Enquiry functions, `isDigit()`, `isODigit()`, Letter

Iteration statement (Definition)

A block of repeating code –a loop.

Availability:	ECMAScript edition –2

An iteration statement executes a body of code iteratively –that is, over and over again, while a certain condition is `true` or until a terminating condition becomes `true`.

An iteration statement consists of a header and a body. The header is a keyword and a set of control parameters in parentheses. The body is a statement block enclosed in braces.

The following iteration statements are supported by ECMA compliant implementations:

- `while()`
- `for ()`
- `for( ... in)`

All of these loop-controlling statements test the condition each time round the loop. In the case of some iterators, the test happens before the code is executed. For others it happens after the code has been executed.

An infinite loop is one for which the terminating condition never happens or where the iterating condition remains `true` forever.

Here are some infinite loops:

```
for(;;){// do this forever}while(true){// do this forever}
```

You can break out of a loop with a break or return. The return statement should only be used when the loop is contained in a called function body. You can call the next iteration prematurely with a continue statement.

Warnings:

- That statement about enclosing code in braces sometimes elicits arguments from developers who routinely prefer to omit braces when the code block consists of a single line.
- However, I stand by the assertion that putting in the braces tends to result in fewer misunderstandings when another programmer is maintaining your source. On the whole I can see no advantages in leaving out the enclosing braces even when the code block is a single line. It helps to remind you that the code is part of an iteration or condition and allows you to add a second line without worrying about the braces.
- In a rush to add a second line, you might very easily omit the braces. That would cause the added line to be executed outside of the iteration loop, whether the condition matched or not. Because of the way people indent braces it is hard to spot when this has happened.

See also:	`break`, `continue`, `for( ... ) ...`, `for( ... in ... ) ...`, `return`, Statement, `while( ... ) ...`

Cross-references:

ECMA 262 edition 2 –section –12.6

ECMA 262 edition 3 –section –12.6

Wrox *Instant JavaScript* –page –25

.jar (File extension)

Java archive file.

JavaScript include files normally have a `.js` file extension. In some rare cases, a `.jar` file may be used. This is really a Java archive file type.

It contains a collection of `.js` files and is normally requested with the `ARCHIVE` tag attribute in the `<SCRIPT>` tag.

A `.jar` file is a ZIP compressed archive of the `.js` files with a small amount of additional information that carries a manifest. The web browser extracts the manifest so that it can then index the archive to retrieve the items it needs. You can make a simple archive like this with a zip compression utility (e.g. Winzip). There is nothing special about it other than making sure the names are consistent with what you refer to from a script or HTML document.

Archives are also useful for attaching digital signatures to scripts. The web reference points to a Netscape resource on archiving, including a link to a downloadable Signtool Archive generator tool.

See also:	<SCRIPT ARCHIVE="...">, File extensions, Security policy, Web browser

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 390

Wrox *Instant JavaScript* –page 3

Web-references:

http://developer.netscape.com/software/signedobj/

Java (Definition)

A language developed by Sun Microsystems that is useful for building plugins for web pages.

Java applets can call JavaScript by means of the exception event mechanism.

You can also embed fragments of JavaScript inside Java applets although there are some limitations to what you can accomplish with this.

For this to work, Java and JavaScript both have to be enabled and working. This may require some preference setting. It also only works in Netscape for the time being.

Things are slightly different with Netscape 6.0. No longer do you get a Java VM packed and embedded in the browser. Instead, it depends on the one you have installed in the OS. If you don't have one installed or if Netscape can't find it, then Java won't work. There's not a great deal of help provided on this aspect of the installation.

Once you do have it installed and working, the Java connectivity should behave as it did before.

Warnings:

- Writing a Java applet to handle a rollover effect is not such a challenging task as you might think, and is one of the things that can be accomplished quite early in your career as a Java programmer. Java is capable of much more than that, but if you are going to tackle a large Java-driven project, you need to think it through and plan it much more carefully.
- Scripting in general seems to be amenable to being developed on the fly and with a minimum of forethought and planning and it's good for putting something together really quickly. Of course getting good at it takes some time. Getting good at Java will probably take longer still. There's a lot more to consider with Java (considerably more than with JavaScript) and the process of creation and deployment is far more unwieldy than putting JavaScript in web pages.
- Your Java code will be far better for having thought the object model through and constructed it carefully. JavaScript lets you get right in there and start manipulating a pretty good object model right away. Of course there are objects already there in Java, but you do need to start building from a lot lower down.

See also:	Java exception events, JavaScript embedded in Java, `Plugin` object

Cross-references:

Wrox *Instant JavaScript* –page 56

Wrox *Professional JavaScript* –page 542

.java (File extension)

Java source file.

A very obscure situation is when JavaScript is hidden inside Java source code. In that case it will be contained in a file with a `.java` extension.

See also:	File extensions, Web browser

Cross-references:

Wrox *Instant JavaScript* –page 3

Java calling JavaScript (Definition)

Java has mechanisms for calling back to JavaScript directly.

For Java to call JavaScript functionality, it needs to have access to the `netscape.javascript.JSObject` class. This is a Java wrapper that encapsulates a JavaScript object and makes it accessible to the Java environment.

Warnings:

- This is only available in Netscape and there is presently no equivalent capability in MSIE.

See also:	`JSObject` object, LiveConnect, `netscape.javascript.JSObject`

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 568

Wrox *Professional JavaScript* –page 544

Java exception events (Definition)

Exceptions in Java applets can trigger JavaScript code in the containing web page.

A Java exception can be coupled to JavaScript handlers using the Netscape BeanConnect technology built into Netscape 4.0 browsers.

This can be used without a great deal of effort on the part of the Java applet programmer. A Java exception could cause a trigger for a call-back event at any time. This may be a way to develop more sophisticated scheduling machines although if you are going to that much trouble, you might choose to implement the bulk of your functionality in the applet itself.

However, this may yield a solution to the portability issues with media players. Given that the providers of the different media players generally provide an API, you may be able to hide the complexity of the different players by embedding the video inside an applet. This would allow you to normalize the JavaScript API it presents to the outside worlds and also deal with any cross-platform problems.

The applet becomes the player but deals with all the differences between different video players and all the nasty cross-platform stuff gets dealt with by the JVM. It's certainly feasible to embed QuickTime inside Java and there also a kit for doing the same with Real Player. Windows media player may be more tricky.

See also:	Event, Java, JavaScript embedded in Java

Cross-references:

Wrox *Instant JavaScript* –page 56

Wrox *Professional JavaScript* –page 549

Java method calls (Definition)

JavaScript allows Java methods to be called natively.

Access to the methods of Java objects is very simple. They appear to be native JavaScript methods. To all intents and purposes, they are no different to the other built-in objects such as `Math`, `Boolean`, `String`, `Date` etc.

There is a subtle difference between the two environments in that JavaScript methods are forgiving about the number and type of arguments whereas Java is not. There is also some ambiguity regarding data types when calling Java methods. It is possible to overload a method in Java and provide alternative interfaces to support different data types. JavaScript can only cope with calling the first interface it encounters and cannot select an appropriate one based on the data type of the method arguments.

You may be able to limit the problems here by carefully considering the data types and coercing them in the script.

Warnings:

- Netscape 3.0 supported Java method calls with a special `JavaMethod` object that was deprecated as of Netscape 4.0.

See also:	LiveConnect

Cross-references:

Wrox *Professional JavaScript* –page 537

Java method data conversion (Definition)

Data types are mapped and converted between JavaScript and Java when methods are called.

When Java methods are called from JavaScript in Netscape they are translated via LiveConnect so that the Java objects appear to be as much like JavaScript objects as possible.

When methods are called, often you want to pass some data to the method. JavaScript supports a variety of primitive data types and more complex objects. These need to be mapped usefully to their corresponding Java equivalent values and types. A similar thing happens when the methods return a result value.

There is some hysteresis effect here because there are not always direct equivalents and so it is possible for the data type to mutate across this boundary, even if the method does nothing other than to echo an input value without modifying it.

It simplifies things a great deal to consider the input arguments and returned values as a separate issue although many factors about the data conversion are similar. On the one hand, we are converting JavaScript values to their nearest and most sensible Java equivalents. On the other hand, we are converting Java values to their best possible JavaScript counterparts.

See also:	Java to JavaScript values, JavaScript to Java values, LiveConnect

Java to JavaScript values (Definition)

Conversion of Java primitives and objects to JavaScript compatible types.

When a Java method returns a value to JavaScript, the Java values need to be converted to compatible and useful data types. This is accomplished with LiveConnect in Netscape Navigator and ActiveX in MSIE.

In some cases a single Java primitive data type will be compatible with JavaScript primitives and objects of many different types. An example of this is the JavaScript Number primitive that will readily accept half a dozen different Java numeric and logical types.

Some Java data types correspond exactly to a single and specific JavaScript type. Examples of this are `null`, `Boolean` and `void`.

A third possibility is that many different Java object types will become a single JavaScript object type. In fact Java objects all become a generic JavaObject.

Here is a table that summarizes the correspondence between Java and JavaScript types when converting to Java. This would be used when passing values into methods under control of JavaScript:

Java	JavaScript
boolean	boolean primitive
byte	number primitive
char	number primitive
double	number primitive
float	number primitive
int	number primitive
long	number primitive

Table continued on following page

Java	JavaScript
null	null
short	number primitive
void	undefined
java.lang.Boolean	JavaObject object
java.lang.Character	JavaObject object
java.lang.Class	JavaObject object
java.lang.Double	JavaObject object
java.lang.Float	JavaObject object
java.lang.Integer	JavaObject object
java.lang.Long	JavaObject object
java.lang.String	JavaObject object
All Java arrays regardless of what they contain	JavaArray
netscape.javascript.JSObject	generic JavaScript object
All other Java objects	JavaObject object

When this happens under the control of Java, there are further limitations as to what can be exchanged between the environments. This is discussed under the individual methods that are affected in the JSObject description.

This table summarizes the relationships at the method return interface. JavaScript values may need to be converted further if they are assigned to JavaScript `LValues`.

JavaScript objects are wrapped so that Java can access them as a `netscape.javascript.JSObject` object. When they are passed back to JavaScript, the encapsulation is removed and the intrinsic JavaScript object is accessed directly again.

Warnings:

- Be aware that you can lose precision when passing data into and out of a Java applet method. As well as type conversion, value truncation may occur. You may not get out what you put in –in terms of accuracy. An example of this is passing a number from JavaScript to a short primitive data type inside Java and back again.
- JavaScript number values can interface to `java.lang.Double` method arguments but will be mapped back to a JavaObject. You may need to extract the numeric value to yield a number primitive again.
- Because JavaScript does not have a character data type, the Java `char` values will be converted to number primitives. You will need to convert them to strings yourself by converting the character's numeric value to a character with the `String.fromCharCode()` method.
- Although number primitives in JavaScript can convert to a `java.lang.Double` value, converting back again yields a JavaObject and not a number primitive. You will need to extract the numeric value again manually.
- the `java.lang.String` object becomes a JavaObject. This means the `String` and `netscape.javascript.JSObject` items mutate as they pass into Java and back again. You may need to do additional work to get the string value back out in a form that you need.
- All Java arrays are returned in a `JavaArray` object. This means that arrays of strings, numbers and objects all appear to be the same from JavaScript and you may need to provide special code to unwrap them properly.

See also:	Java method data conversion, `java.lang.Boolean`, `java.lang.Character`, `java.lang.Class`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.String`, `JavaArray` object, `JavaObject` object, JavaScript to Java values, `JSObject` object, `JSObject.call()`, `JSObject.setMember()`, `JSObject.setSlot()`, LiveConnect, `Math` object, `netscape.javascript.JSObject`, `String.fromCharCode()`

Cross-references:

Wrox *Professional JavaScript* –page 533

java (Property)

A short cut reference to the `Packages.java` object.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`JavaPackage java`	
JavaScript syntax:	N	`java`
	N	*`myWindow`*`.java`
	N	*`myWindow`*`.Packages.java`
	N	`Packages.java`

See also:	`java.awt`, `Window.java`, `Window.Packages`

Property attributes:

`ReadOnly`.

Refer to:

`Packages`.java

java.awt (Java package)

A JavaPackage that represents the Abstract Windowing Toolkit.

Availability:	JavaScript –1.1 Netscape –3.0

The Abstract Windowing Toolkit is a set of Classes that provide a platform-independent way to develop graphical user interfaces on your applets or Java applications. It was introduced quite early on in the Java history at version 1.0 and had some major additions at Java 1.1, and it has now been supplemented by the Swing components which override some AWT functionality but do not render it entirely obsolete.

java.awt.Button (Java class)

A JavaClass object that represents the `java.awt.Button` class.

Availability:	JavaScript –1.1 Netscape –3.0
Object methods:	`action(), addNotify(), bounds(), checkImage(), createImage(), deliverEvent(), disable(), enable(), equals(), getBackground(), getClass(), getColorModel(), getFont(), getFontMetrics(), getForeground(), getGraphics(), getLabel(), getLocale(), getParent(), getPeer(), getToolkit(), gotFocus(), hashCode(), hide(), imageUpdate(), inside(), invalidate(), isEnabled(), isShowing(), isValid(), isVisible(), keyDown(), keyUp(), layout(), list(), locate(), location(), lostFocus(), minimumSize(), mouseDown(), mouseDrag(), mouseEnter(), mouseExit(), mouseMove(), mouseUp(), move(), nextFocus(), notify(), notifyAll(), paint(), paintAll(), postEvent(), preferredSize(), prepareImage(), print(), printAll(), removeNotify(), repaint(), requestFocus(), reshape(), resize(), setBackground(), setFont(), setForeground(), setLabel(), setVisible(), show(), size(), toString(), update(), validate(), wait()`

This is a sub-class of the AWT Component Class. It describes a button labelled with a text legend which the user can click on. Clicking on the button initiates an action. Each button has an action associated with it and when it is clicked, it will generate an `ActionEvent` object.

Method	JavaScript	JScript	N	IE	Opera	Notes
`action()`	1.1 +	-	3.0 +	-	-	-
`addNotify()`	1.1 +	-	3.0 +	-	-	-
`bounds()`	1.1 +	-	3.0 +	-	-	-
`checkImage()`	1.1 +	-	3.0 +	-	-	-
`createImage()`	1.1 +	-	3.0 +	-	-	-
`deliverEvent()`	1.1 +	-	3.0 +	-	-	-
`disable()`	1.1 +	-	3.0 +	-	-	-
`enable()`	1.1 +	-	3.0 +	-	-	-
`equals()`	1.1 +	-	3.0 +	-	-	-
`getBackground()`	1.1 +	-	3.0 +	-	-	-
`getClass()`	1.1 +	-	3.0 +	-	-	-
`getColorModel()`	1.1 +	-	3.0 +	-	-	-

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	Notes
getFont()	1.1 +	-	3.0 +	-	-	-
getFontMetrics()	1.1 +	-	3.0 +	-	-	-
getForeground()	1.1 +	-	3.0 +	-	-	-
getGraphics()	1.1 +	-	3.0 +	-	-	-
getLabel()	1.1 +	-	3.0 +	-	-	-
getLocale()	1.1 +	-	3.0 +	-	-	-
getParent()	1.1 +	-	3.0 +	-	-	-
getPeer()	1.1 +	-	3.0 +	-	-	-
getToolkit()	1.1 +	-	3.0 +	-	-	-
gotFocus()	1.1 +	-	3.0 +	-	-	-
hashCode()	1.1 +	-	3.0 +	-	-	-
hide()	1.1 +	-	3.0 +	-	-	-
imageUpdate()	1.1 +	-	3.0 +	-	-	-
inside()	1.1 +	-	3.0 +	-	-	-
invalidate()	1.1 +	-	3.0 +	-	-	-
isEnabled()	1.1 +	-	3.0 +	-	-	-
isShowing()	1.1 +	-	3.0 +	-	-	-
isValid()	1.1 +	-	3.0 +	-	-	-
isVisible()	1.1 +	-	3.0 +	-	-	-
keyDown()	1.1 +	-	3.0 +	-	-	-
keyUp()	1.1 +	-	3.0 +	-	-	-
layout()	1.1 +	-	3.0 +	-	-	-
list()	1.1 +	-	3.0 +	-	-	-
locate()	1.1 +	-	3.0 +	-	-	-
location()	1.1 +	-	3.0 +	-	-	-
lostFocus()	1.1 +	-	3.0 +	-	-	-
minimumSize()	1.1 +	-	3.0 +	-	-	-
mouseDown()	1.1 +	-	3.0 +	-	-	-
mouseDrag()	1.1 +	-	3.0 +	-	-	-
mouseEnter()	1.1 +	-	3.0 +	-	-	-
mouseExit()	1.1 +	-	3.0 +	-	-	-
mouseMove()	1.1 +	-	3.0 +	-	-	-
mouseUp()	1.1 +	-	3.0 +	-	-	-
move()	1.1 +	-	3.0 +	-	-	-
nextFocus()	1.1 +	-	3.0 +	-	-	-
notify()	1.1 +	-	3.0 +	-	-	-
notifyAll()	1.1 +	-	3.0 +	-	-	-

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	Notes
`paint()`	1.1 +	-	3.0 +	-	-	-
`paintAll()`	1.1 +	-	3.0 +	-	-	-
`postEvent()`	1.1 +	-	3.0 +	-	-	-
`preferredSize()`	1.1 +	-	3.0 +	-	-	-
`prepareImage()`	1.1 +	-	3.0 +	-	-	-
`print()`	1.1 +	-	3.0 +	-	-	-
`printAll()`	1.1 +	-	3.0 +	-	-	-
`removeNotify()`	1.1 +	-	3.0 +	-	-	-
`repaint()`	1.1 +	-	3.0 +	-	-	-
`requestFocus()`	1.1 +	-	3.0 +	-	-	-
`reshape()`	1.1 +	-	3.0 +	-	-	-
`resize()`	1.1 +	-	3.0 +	-	-	-
`setBackground()`	1.1 +	-	3.0 +	-	-	-
`setFont()`	1.1 +	-	3.0 +	-	-	-
`setForeground()`	1.1 +	-	3.0 +	-	-	-
`setLabel()`	1.1 +	-	3.0 +	-	-	-
`setVisible()`	1.1 +	-	3.0 +	-	-	-
`show()`	1.1 +	-	3.0 +	-	-	-
`size()`	1.1 +	-	3.0 +	-	-	-
`toString()`	1.1 +	-	3.0 +	-	-	-
`update()`	1.1 +	-	3.0 +	-	-	-
`validate()`	1.1 +	-	3.0 +	-	-	-
`wait()`	1.1 +	-	3.0 +	-	-	-

java.awt.image (Java class)

A JavaClass object that represents the `java.awt.image` class.

Availability:	JavaScript –1.1 Netscape –3.0

This is an abstract class provided as the super-class of all classes that describe and encapsulate images within AWT. Because it is an abstract class, the methods are mostly place holders and will need to be overridden in the concrete classes that are sub-classed from this class.

java.lang (Java package)

The Java language package.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`JavaPackage object`	
JavaScript syntax:	N	`java.lang`

This JavaPackage provides the main language support classes for Java. There are classes for handling variables, multi-threaded programming, system classes, string-handling and error/exception handler support.

It's not likely you would want to delve deeply into these issues, but occasionally you may need to enquire of some readable attributes of a Java applet and this package may help.

java.lang.Boolean (Java class)

The Java Boolean class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class properties:	`FALSE, TRUE, TYPE`
Class methods:	`getBoolean(), valueOf()`
Object methods:	`booleanValue(), equals(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait()`

Support for Boolean primitives in Java.

Values of this type are visible to JavaScript as a JavaObject, which encapsulates the Java created value.

See also:	boolean, Java to JavaScript values, JavaScript to Java values

Method	JavaScript	JScript	N	IE	Opera	Notes
`booleanValue()`	1.1 +	-	3.0 +	-	-	-
`equals()`	1.1 +	-	3.0 +	-	-	-
`getClass()`	1.1 +	-	3.0 +	-	-	-
`hashCode()`	1.1 +	-	3.0 +	-	-	-
`notify()`	1.1 +	-	3.0 +	-	-	-
`notifyAll()`	1.1 +	-	3.0 +	-	-	-
`toString()`	1.1 +	-	3.0 +	-	-	-
`wait()`	1.1 +	-	3.0 +	-	-	-

java.lang.Character (Java class)

The Java Character class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class properties:	`COMBINING_SPACING_MARK, CONNECTOR_PUNCTUATION, CONTROL, CURRENCY_SYMBOL, DASH_PUNCTUATION, DECIMAL_DIGIT_NUMBER, ENCLOSING_MARK, END_PUNCTUATION, FORMAT, LETTER_NUMBER, LINE_SEPARATOR, LOWERCASE_LETTER, MATH_SYMBOL, MAX_RADIX, MAX_VALUE, MIN_RADIX, MIN_VALUE, MODIFIER_LETTER, MODIFIER_SYMBOL, NON_SPACING_MARK, OTHER_LETTER, OTHER_NUMBER, OTHER_PUNCTUATION, OTHER_SYMBOL, PARAGRAPH_SEPARATOR, PRIVATE_USE, SPACE_SEPARATOR, START_PUNCTUATION, SURROGATE, TITLECASE_LETTER, TYPE, UNASSIGNED, UPPERCASE_LETTER`
Class methods:	`digit(), forDigit(), getNumericValue(), getType(), isDefined(), isDigit(), isIdentifierIgnorable(), isISOControl(), isJavaIdentifierPart(), isJavaIdentifierStart(), isJavaLetter(), isJavaLetterOrDigit(), isLetter(), isLetterOrDigit(), isLowerCase(), isSpace(), isSpaceChar(), isTitleCase(), isUnicodeIdentifierPart(), isUnicodeIdentifierStart(), isUpperCase(), isWhitespace(), toLowerCase(), toTitleCase(), toUpperCase()`
Object methods:	`charValue(), equals(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait()`

Support for character based data in the Java environment.

Values of this type are visible to JavaScript as a JavaObject which encapsulates the Java created value.

See also:	`char`, Java to JavaScript values

Method	JavaScript	JScript	N	IE	Opera	Notes
`charValue()`	1.1 +	-	3.0 +	-	-	-
`equals()`	1.1 +	-	3.0 +	-	-	-
`getClass()`	1.1 +	-	3.0 +	-	-	-
`hashCode()`	1.1 +	-	3.0 +	-	-	-
`notify()`	1.1 +	-	3.0 +	-	-	-
`notifyAll()`	1.1 +	-	3.0 +	-	-	-
`toString()`	1.1 +	-	3.0 +	-	-	-
`wait()`	1.1 +	-	3.0 +	-	-	-

java.lang.Class (Java class)

The Java Class class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class methods:	`forName()`

Support for access to the Java Classes that define the structure of objects.

Values of this type are visible to JavaScript as a JavaObject which encapsulates the Java created value.

You cannot instantiate new objects of this class because the constructor is not public.

See also:	class, Java to JavaScript values

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 358

java.lang.Double (Java class)

The Java numeric double precision class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class properties:	`MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY, TYPE`
Class methods:	`doubleToLongBits(), isInfinite(), isNaN(), longBitsToDouble(), toString(), valueOf()`
Object methods:	`byteValue(), doubleValue(), equals(), floatValue(), getClass(), hashCode(), intValue(), isInfinite(), isNaN(), longValue(), notify(), notifyAll(), shortValue(), toString(), wait()`

Support for double precision floating point numbers.

Values of this type are visible to JavaScript as a JavaObject which encapsulates the Java created value.

See also:	`double`, Java to JavaScript values, JavaScript to Java values

Method	JavaScript	JScript	N	IE	Opera	Notes
byteValue()	1.1 +	-	3.0 +	-	-	-
doubleValue()	1.1 +	-	3.0 +	-	-	-
equals()	1.1 +	-	3.0 +	-	-	-
floatValue()	1.1 +	-	3.0 +	-	-	-
getClass()	1.1 +	-	3.0 +	-	-	-
hashCode()	1.1 +	-	3.0 +	-	-	-
intValue()	1.1 +	-	3.0 +	-	-	-
isInfinite()	1.1 +	-	3.0 +	-	-	-
isNaN()	1.1 +	-	3.0 +	-	-	-
longValue()	1.1 +	-	3.0 +	-	-	-
notify()	1.1 +	-	3.0 +	-	-	-
notifyAll()	1.1 +	-	3.0 +	-	-	-
shortValue()	1.1 +	-	3.0 +	-	-	-
toString()	1.1 +	-	3.0 +	-	-	-
wait()	1.1 +	-	3.0 +	-	-	-

java.lang.Float (Java class)

The Java Floating point numeric class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class properties:	`MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY, TYPE`
Class methods:	`floatToIntBits(), intBitsToFloat(), isInfinite(), isNaN(), toString(), valueOf()`
Object methods:	`byteValue(), doubleValue(), equals(), floatValue(), getClass(), hashCode(), intValue(), isInfinite(), isNaN(), longValue(), notify(), notifyAll(), shortValue(), toString(), wait()`

Support for single precision floating point numbers.

Values of this type are visible to JavaScript as a JavaObject, which encapsulates the Java created value.

See also:	`float`, Java to JavaScript values

Method	JavaScript	JScript	N	IE	Opera	Notes
byteValue()	1.1 +	-	3.0 +	-	-	-
doubleValue()	1.1 +	-	3.0 +	-	-	-
equals()	1.1 +	-	3.0 +	-	-	-
floatValue()	1.1 +	-	3.0 +	-	-	-
getClass()	1.1 +	-	3.0 +	-	-	-
hashCode()	1.1 +	-	3.0 +	-	-	-
intValue()	1.1 +	-	3.0 +	-	-	-
isInfinite()	1.1 +	-	3.0 +	-	-	-
isNaN()	1.1 +	-	3.0 +	-	-	-
longValue()	1.1 +	-	3.0 +	-	-	-
notify()	1.1 +	-	3.0 +	-	-	-
notifyAll()	1.1 +	-	3.0 +	-	-	-
shortValue()	1.1 +	-	3.0 +	-	-	-
toString()	1.1 +	-	3.0 +	-	-	-
wait()	1.1 +	-	3.0 +	-	-	-

java.lang.Integer (Java class)

The Java integer numeric class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class properties:	`MAX_VALUE, MIN_VALUE, TYPE`
Class methods:	`decode(), getInteger(), parseInt(), toBinaryString(), toHexString(), toOctalString(), toString(), valueOf()`
Object methods:	`byteValue(), doubleValue(), equals(), floatValue(), getClass(), hashCode(), intValue(), longValue(), notify(), notifyAll(), shortValue(), toString(), wait()`

Support for integer values in the Java environment.

Values of this type are visible to JavaScript as a JavaObject which encapsulates the Java created value.

See also:	`int`, Java to JavaScript values

Method	JavaScript	JScript	N	IE	Opera	Notes
byteValue()	1.1 +	-	3.0 +	-	-	-
doubleValue()	1.1 +	-	3.0 +	-	-	-
equals()	1.1 +	-	3.0 +	-	-	-
floatValue()	1.1 +	-	3.0 +	-	-	-
getClass()	1.1 +	-	3.0 +	-	-	-
hashCode()	1.1 +	-	3.0 +	-	-	-
intValue()	1.1 +	-	3.0 +	-	-	-
longValue()	1.1 +	-	3.0 +	-	-	-
notify()	1.1 +	-	3.0 +	-	-	-
notifyAll()	1.1 +	-	3.0 +	-	-	-
shortValue()	1.1 +	-	3.0 +	-	-	-
toString()	1.1 +	-	3.0 +	-	-	-
wait()	1.1 +	-	3.0 +	-	-	-

java.lang.Long (Java class)

The Java long numeric class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class properties:	`MAX_VALUE, MIN_VALUE, TYPE`
Class methods:	`getLong(), parseLong(), toBinaryString(), toHexString(), toOctalString(), toString(), valueOf()`
Object methods:	`byteValue(), doubleValue(), equals(), floatValue(), getClass(), hashCode(), intValue(), longValue(), notify(), notifyAll(), shortValue(), toString(), wait()`

Support for long integer values in the Java environment.

Values of this type are visible to JavaScript as a JavaObject, which encapsulates the Java created value.

See also:	Java to JavaScript values, `long`

Method	JavaScript	JScript	N	IE	Opera	Notes
byteValue()	1.1 +	-	3.0 +	-	-	-
doubleValue()	1.1 +	-	3.0 +	-	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
equals()	1.1 +	-	3.0 +	-	-	-
floatValue()	1.1 +	-	3.0 +	-	-	-
getClass()	1.1 +	-	3.0 +	-	-	-
hashCode()	1.1 +	-	3.0 +	-	-	-
intValue()	1.1 +	-	3.0 +	-	-	-
longValue()	1.1 +	-	3.0 +	-	-	-
notify()	1.1 +	-	3.0 +	-	-	-
notifyAll()	1.1 +	-	3.0 +	-	-	-
shortValue()	1.1 +	-	3.0 +	-	-	-
toString()	1.1 +	-	3.0 +	-	-	-
wait()	1.1 +	-	3.0 +	-	-	-

java.lang.Object (Java class)

This is the super-class of all objects in the Java `lang` environment.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Object methods:	`equals(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait()`

Support for generic objects in the Java environment.

Methods and properties belonging to this object are inherited by objects such as the `JavaArray` object (as it appears to JavaScript).

This class is presented to the JavaScript programmer as a JavaObject object.

See also:	JavaObject object

Method	JavaScript	JScript	N	IE	Opera	Notes
equals()	1.1 +	-	3.0 +	-	-	-
getClass()	1.1 +	-	3.0 +	-	-	-
hashCode()	1.1 +	-	3.0 +	-	-	-
notify()	1.1 +	-	3.0 +	-	-	-
notifyAll()	1.1 +	-	3.0 +	-	-	-
toString()	1.1 +	-	3.0 +	-	-	-
wait()	1.1 +	-	3.0 +	-	-	-

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 566

java.lang.String (Java class)

The Java String class.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`JavaObject object`
Class methods:	`copyValueOf(), valueOf()`
Object methods:	`charAt(), compareTo(), concat(), endsWith(), equals(), equalsIgnoreCase(), getBytes(), getChars(), getClass(), hashCode(), indexOf(), intern(), lastIndexOf(), length(), notify(), notifyAll(), regionMatches(), replace(), startsWith(), substring(), toCharArray(), toLowerCase(), toString(), toUpperCase(), trim(), wait()`

Support for string values in the Java environment.

Values of this type are visible to JavaScript as a JavaObject, which encapsulates the Java created value.

See also:	Java to JavaScript values, JavaScript to Java values, `String`

Method	JavaScript	JScript	N	IE	Opera	Notes
`charAt()`	1.1 +	-	3.0 +	-	-	-
`compareTo()`	1.1 +	-	3.0 +	-	-	-
`concat()`	1.1 +	-	3.0 +	-	-	-
`endsWith()`	1.1 +	-	3.0 +	-	-	-
`equals()`	1.1 +	-	3.0 +	-	-	-
`equalsIgnoreCase()`	1.1 +	-	3.0 +	-	-	-
`getBytes()`	1.1 +	-	3.0 +	-	-	-
`getChars()`	1.1 +	-	3.0 +	-	-	-
`getClass()`	1.1 +	-	3.0 +	-	-	-
`hashCode()`	1.1 +	-	3.0 +	-	-	-
`indexOf()`	1.1 +	-	3.0 +	-	-	-
`intern()`	1.1 +	-	3.0 +	-	-	-
`lastIndexOf()`	1.1 +	-	3.0 +	-	-	-
`length()`	1.1 +	-	3.0 +	-	-	-
`notify()`	1.1 +	-	3.0 +	-	-	-
`notifyAll()`	1.1 +	-	3.0 +	-	-	-
`regionMatches()`	1.1 +	-	3.0 +	-	-	-
`replace()`	1.1 +	-	3.0 +	-	-	-

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	Notes
startsWith()	1.1 +	-	3.0 +	-	-	-
substring()	1.1 +	-	3.0 +	-	-	-
toCharArray()	1.1 +	-	3.0 +	-	-	-
toLowerCase()	1.1 +	-	3.0 +	-	-	-
toString()	1.1 +	-	3.0 +	-	-	-
toUpperCase()	1.1 +	-	3.0 +	-	-	-
trim()	1.1 +	-	3.0 +	-	-	-
wait()	1.1 +	-	3.0 +	-	-	-

java.util (Java package)

The Java utility package.

Availability:	JavaScript –1.1 Netscape –3.0

This package provides a collection of classes and interfaces that are generally useful to the programmer. These fall into the following categories:

- Arrays
- Calender functions and date support
- Collections
- Event objects
- Mapping classes
- Random numbers
- Resource bundle support
- String tokenizer
- Timers
- Timer tasks

See also:	Method, Property

java.util.Date (Java class)

The Java Date object.

Availability:	JavaScript –1.1 Netscape –3.0
Class methods:	parse(), UTC()

Object methods:	`after()`, `before()`, `equals()`, `getClass()`, `getDate()`, `getDay()`, `getHours()`, `getMinutes()`, `getMonth()`, `getSeconds()`, `getTime()`, `getTimezoneOffset()`, `getYear()`, `hashCode()`, `notify()`, `notifyAll()`, `setDate()`, `setHours()`, `setMinutes()`, `setMonth()`, `setSeconds()`, `setTime()`, `setYear()`, `toGMTString()`, `toLocaleString()`, `toString()`, `wait()`

This class is an encapsulation of the machine date and time measured in milliseconds since January the 1st, 1970. Fortunately, this is the same reference time that the JavaScript standard (ECMA) prescribes.

See also:	Date object, Universal coordinated time

Method	JavaScript	JScript	N	IE	Opera	Notes
`after()`	1.1 +	-	3.0 +	-	-	-
`before()`	1.1 +	-	3.0 +	-	-	-
`equals()`	1.1 +	-	3.0 +	-	-	-
`getClass()`	1.1 +	-	3.0 +	-	-	-
`getDate()`	1.1 +	-	3.0 +	-	-	-
`getDay()`	1.1 +	-	3.0 +	-	-	-
`getHours()`	1.1 +	-	3.0 +	-	-	-
`getMinutes()`	1.1 +	-	3.0 +	-	-	-
`getMonth()`	1.1 +	-	3.0 +	-	-	-
`getSeconds()`	1.1 +	-	3.0 +	-	-	-
`getTime()`	1.1 +	-	3.0 +	-	-	-
`getTimezoneOffset()`	1.1 +	-	3.0 +	-	-	-
`getYear()`	1.1 +	-	3.0 +	-	-	-
`hashCode()`	1.1 +	-	3.0 +	-	-	-
`notify()`	1.1 +	-	3.0 +	-	-	-
`notifyAll()`	1.1 +	-	3.0 +	-	-	-
`setDate()`	1.1 +	-	3.0 +	-	-	-
`setHours()`	1.1 +	-	3.0 +	-	-	-
`setMinutes()`	1.1 +	-	3.0 +	-	-	-
`setMonth()`	1.1 +	-	3.0 +	-	-	-
`setSeconds()`	1.1 +	-	3.0 +	-	-	-
`setTime()`	1.1 +	-	3.0 +	-	-	-
`setYear()`	1.1 +	-	3.0 +	-	-	-
`toGMTString()`	1.1 +	-	3.0 +	-	-	-
`toLocaleString()`	1.1 +	-	3.0 +	-	-	-
`toString()`	1.1 +	-	3.0 +	-	-	-
`wait()`	1.1 +	-	3.0 +	-	-	-

JavaArray object (Object/Navigator)

A JavaScript data type that encapsulates a Java array.

Availability:	JavaScript –1.1 Netscape –3.0	
JavaScript syntax:	N	*myJavaArray = myWindow.Packages*
Object properties:	`length`	
Object methods:	`toString()`	

In Java, objects can be collected together into arrays. This is also true of JavaScript. However, as you might expect, some encapsulation of the Java array is necessary to be able to operate on it from a JavaScript environment. The `JavaArray` object does just this. It supports some JavaScript-like behavior in that it has a length property and can be accessed element by element using the JavaScript `[]` notation. JavaArray objects can be stacked to make multidimensional arrays and they can also be traversed with a `for( ... in ... ) ...` loop.

At version 1.4 of JavaScript, a `JavaArray` object inherits properties from the `java.lang.Object` super-class.

See also:	`Array` object, `Collection` object, Java to JavaScript values, JavaScript to Java values, LiveConnect, `Window.Packages`

Property	JavaScript	JScript	N	IE	Opera	Notes
`length`	1.1 +	-	3.0 +	-	-	ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	Notes
`toString()`	1.1 +	-	3.0 +	-	-	-

JavaArray.length (Property)

The length of the Java array encapsulated by the `JavaArray` object.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	`myJavaArray.length`

See also:	LiveConnect, `Collection.length`

Property attributes:

`ReadOnly`.

JavaArray.toString() (Method)

A string primitive representation of the JavaArray contents.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*`myJavaArray`*`.toString()`

In version 1.4 of JavaScript, this method is overridden by the `toString()` method belonging to the `java.lang.Object` super-class.

Prior to that, it yielded a string like this:

```
[object JavaArray]
```

If you expect to use this for any useful purpose in scripting, you will need to override it with a custom handler that converts the object to a string that delivers the instance value rather than the type of its prototype object.

JavaClass object (Object/Navigator)

A JavaScript data type that encapsulates a Java Class.

Availability:	JavaScript –1.1 Netscape –3.0	
JavaScript syntax:	N	*`myJavaClass = myWindow`*`.Packages.java`

The static fields of the Java Class are presented as properties of the JavaClass object.

The static methods of the Java Class are presented as methods available to the JavaClass object. They may be presented as `JavaMethod` objects depending on how you access them from JavaScript.

The only properties and methods this object has, correspond to the public properties and methods of the Java class that it represents. You can enumerate these properties in a loop.

Java classes are either stored in individual class files or are collected together into groups stored in a ZIP file. Groups of classes will be represented by a `JavaPackage` object. Each individual class is represented by a JavaClass object. The JavaClass objects belong to a parent `JavaPackage` object.

To operate on these objects, you really need to know something about the Java classes they encapsulate. There are many standard classes and some that may have been custom written for your project.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Create a JavaScript object that encapsulates a Java Class
var myJavaDateClass  = Packages.java.util.Date;
var myJavaDateObject = new Packages.java.util.Date;
// Write the date to a web page
document.write(myJavaDateObject.toString());
// Write the same date value date to the Java console
java.lang.System.out.println(myJavaDateObject);
</SCRIPT>
</BODY>
</HTML>
```

See also:	JavaScript to Java values, LiveConnect, `Packages.java`, `Packages.netscape`, `Window.Packages`

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 358

JavaMethod object (Object/Navigator)

A deprecated way of encapsulating method calls on Java objects from JavaScript.

Availability:	JavaScript –1.1 Netscape –3.0 Deprecated	
JavaScript syntax:	N	*myJavaMethod = myJavaObject.aMethod()*
Argument list:	*aMethod*	A method belonging to the Java object

See also:	JavaScript to Java values, Window.Packages, Java method calls

Cross-references:

Wrox *Professional JavaScript* –page 527

JavaObject object (Object/Navigator)

A JavaScript data type that encapsulates a Java object. These are generally going to be members of the Java component class

Availability:	JavaScript –1.1 Netscape –3.0	
JavaScript syntax:	N	*myJavaObject* = new java.lang.Object
	N	*myJavaObject* = document.applets[*anIndex*];
Object properties:	description, filename, length, name	
Object methods:	booleanValue(), destroy(), disable(), doubleValue(), enable(), getAppletContext(), getAppletInfo(), getBackground(), getClass(), getCodeBase(), getDocumentBase(), getLocale(), getParameter(), getParameterInfo(), getToolkit(), hide(), init(), isActive(), isEnabled(), isShowing(), isValid(), isVisible(), minimumSize(), refresh(), start(), stop(), toString()	

To make any serious use of this object, you need to know a little Java –at least enough to be able to be familiar with the class structures and creating and modifying objects. If you know how to make your own applets then that is probably sufficient to get started with.

The public properties of the Java Class of which the JavaObject is an instance are presented as properties of the JavaObject object. A JavaObject also inherits the properties from the `java.lang.Object` Class and any other classes which are in its superclass hierarchy. These are generally available by means of accessor methods so they will likely be listed as methods rather than properties.

The public methods of the Java Class of which the JavaObject is an instance are presented as methods of the JavaObject object. A JavaObject also inherits the methods from the `java.lang.Object` Class and any other classes which are in its superclass hierarchy.

The only properties and methods this object has, correspond to the public properties and methods of the Java object that it represents. You can enumerate the properties in a loop to inspect the interface to the object from your own scripts; like this:

```
for(myProp in myJavaObject) {
   document.write(myProp);
   document.write("<BR>");
}
for(myProp in myJavaObject) {
   document.write(myProp);
   document.write("<BR>");
```

There may be some properties that are not revealed by this and you may need to resort to some Java documentation for further details. Properties and methods for the objects in Java are documented in more depth in *Java Programmer's Reference*, written by Grant Palmer and published by Wrox Press, which covers JDK 1.3 extensively.

To operate on the `JavaObject` objects, you really need to know something about the Java objects they encapsulate. There are many standard classes and some that may have been custom written for your project.

Beware that the object may report accessors that don't actually have any purpose in the object you have enumerated.

For properties and methods that apply to the `Applet` object in the context of an MSIE browser, examine the Applet object topic and its related items. This is documented separately because it does not support the same Java –JavaScript bridging mechanism and provides a mutually exclusive set of properties for communication with the `Applet` object.

Although JavaScript exposes a great many of the properties of an Applet by means of `Accessor` Methods, it is probably not safe to call the more destructive of them from JavaScript. However, you may usefully want to use the various enquiry methods to find out about the Applet and its internals. Many of these `Accessor` Methods yield other objects whose properties can also be inspected by using the same enumeration techniques.

Having examined some applets and other miscellaneous Java objects, a summary list of the accessor methods is presented at the head of this topic.

Some fragments of example code are given here and a couple of important methods are described in adjacent topics. Documenting all the interfaces to an Applet or Java object is so dependent on the object that you should refer to the Applet documentation and source and a Java reference manual for details of the internals of Java code.

Warnings:

- When JavaObject objects are used in JavaScript expressions, even though internally they may contain a value that could be represented by a JavaScript primitive type, they do not behave like that JavaScript primitive type. Instead, they behave according to the rules of the Java object type that contains the value. This can sometimes cause scripts to behave in unexpected ways, for example you might get a concatenation instead of a numeric addition. You should also explicitly test for values and not make assumptions that a `null` or undefined value is returned.
- The JavaObject in Netscape is notably unforgiving. Many other JavaScript objects will yield the value undefined if you ask for a property that does not exist. A JavaObject will generate a run time error and so you cannot easily test for the existence of an unknown property.
- Some objects also support a strangely named function that is referred to by an enumerable but apparently unnamed property. It is exposed when you enumerate the properties in a `for( ... in ... )` loop and output their names with a `document.write()`. A property appears in the list that seems to have no name. Checking the property names with the `typeof` operator reports this as a string so it's not an undefined name. OK, so lets measure its length. The suspect property reports a length of 6 in the example I was testing but it is still invisible. Here's where we try hunches and use our debugging instincts. Wrap the property name in the `escape()` function. That should tell us what weird characters are there. Voila. It reports that the name is now `%3CInit%3E`. So the name is the word `Init` wrapped in < and > characters, which explains why we couldn't see it. The web browser thought it was a tag and didn't recognize it so it was hidden.
- The second piece of example code shows how the properties were enumerated and how to display the hidden property name.

Example code:

```
// Output some text to the Java console
java.lang.System.out.println("Some text message");
// Create a JavaScript object that encapsulates a Java Objectvar
```

```
myJavaDateObject = new Packages.java.util.Date;

-----------------------------------------------------------------

<!-- Debugging hidden property values -->
<HTML>
<HEAD>
</HEAD>
<BODY>
<TABLE BORDER=1>
<SCRIPT>
// Create a JavaScript object that encapsulates a Java Class
var myJavaDateClass = new Packages.java.util.Date;
// Now enumerate its properties
var myIndex = 0;
for(myProp in myJavaDateClass)
{
   document.write("<TR><TD>");
   document.write(myIndex);
   document.write("</TD><TD>");
   document.write(myProp);
   document.write("</TD><TD>");
   document.write(typeof(myProp));
   document.write("</TD><TD>");
   document.write(myProp.length);
   document.write("</TD><TD>");
   document.write(escape(myProp));
   document.write("</TD></TR>");

   myIndex++;
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	`Applet` object, `Document.applets[]`, `Document.embeds[]`, `EmbedArray` object, Java to JavaScript values, `java.lang.Object`, `JavaObject.booleanValue()`, JavaScript to Java values, LiveConnect, `Packages.java, Packages.netscape, Window.Packages`

Property	JavaScript	JScript	N	IE	Opera	Notes
`description`	1.1 +	-	3.0 +	-	-	Warning
`filename`	1.1 +	-	3.0 +	-	-	Warning
`length`	1.1 +	-	3.0 +	-	-	Warning
`name`	1.1 +	-	3.0 +	-	-	Warning

Method	JavaScript	JScript	N	IE	Opera	Notes
`booleanValue()`	1.2 +	-	4.0 +	-	-	-
`destroy()`	1.1 +	-	3.0 +	-	-	Warning
`disable()`	1.1 +	-	3.0 +	-	-	Warning
`doubleValue()`	1.1 +	-	3.0 +	-	-	Warning

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	Notes
enable()	1.1 +	-	3.0 +	-	-	Warning
getAppletContext()	1.1 +	-	3.0 +	-	-	Warning
getAppletInfo()	1.1 +	-	3.0 +	-	-	Warning
getBackground()	1.1 +	-	3.0 +	-	-	Warning
getClass()	1.2 +	-	4.0 +	-	-	Warning
getCodeBase()	1.1 +	-	3.0 +	-	-	Warning
getDocumentBase()	1.1 +	-	3.0 +	-	-	Warning
getLocale()	1.1 +	-	3.0 +	-	-	Warning
getParameter()	1.1 +	-	3.0 +	-	-	Warning
getParameterInfo()	1.1 +	-	3.0 +	-	-	Warning
getToolkit()	1.1 +	-	3.0 +	-	-	Warning
hide()	1.1 +	-	3.0 +	-	-	Warning
init()	1.1 +	-	3.0 +	-	-	Warning
isActive()	1.1 +	-	3.0 +	-	-	Warning
isEnabled()	1.1 +	-	3.0 +	-	-	Warning
isShowing()	1.1 +	-	3.0 +	-	-	Warning
isValid()	1.1 +	-	3.0 +	-	-	Warning
isVisible()	1.1 +	-	3.0 +	-	-	Warning
minimumSize()	1.1 +	-	3.0 +	-	-	Warning
refresh()	1.1 +	-	3.0 +	-	-	Warning
start()	1.1 +	-	3.0 +	-	-	Warning
stop()	1.1 +	-	3.0 +	-	-	Warning
toString()	1.1 +	-	3.0 +	-	-	Warning

JavaObject.booleanValue() (Method/Java)

This is the value that is used when the JavaObject is used in a Boolean context.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myJavaObject*.booleanValue()

Java Boolean objects support this method. It is a Java method but it is analogous to the `valueOf()` method used on a JavaScript Boolean object.

Because all Java objects become members of the JavaObject class they respond to `valueOf()` and `toString()` but those can be overridden in the JavaScript environment. The `booleanValue()` method penetrates to the Java environment and is executed there with its result encapsulated. The `valueOf()` and `toString()` methods are executed in the JavaScript environment even if there is a corresponding Java method for each.

The example shows all three methods being invoked but the `toString()` and `valueOf()` are masked. Note that by masking `toString()`, `valueOf()` is affected too. That may not be common to all object types.

There are other similar methods to support different Java primitive data types when they are encapsulated in an object. Only this one is illustrated. The others are different only in the data type and the name they have.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Create a JavaScript object that encapsulates a Java Class
var myJavaBooleanClass = new Packages.java.lang.Boolean(true);
// Mask the JavaScript environment values
myJavaBooleanClass.toString = mask;
// Write the object to a web page
document.write("toString() : ");
document.write(myJavaBooleanClass.toString());
document.write("<BR>");
document.write("valueOf() : ");
document.write(myJavaBooleanClass.valueOf());
document.write("<BR>");
document.write("booleanValue() : ");
document.write(myJavaBooleanClass.booleanValue());
document.write("<BR>");
// Masking function
function mask()
{
return "Masked";
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`JavaObject` object, LiveConnect

JavaObject.getClass() (Method/Java)

A JavaScript method for obtaining the class of a JavaObject.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myJavaObject*`.getClass()`

This returns the Java class of the encapsulated Java object. This is not necessarily the same as a JavaScript class name. In fact what is returned is an object reference to a class object which may need further work to extract the name of the class.

The example shows how a Java object is instantiated and then enquired to as of its class. Using the LiveConnect, the `getClass()` method requests the class name from the Java code and then converts that to a JavaScript message. Inspecting the constructor of the new object and extracting its name property tells you what class JavaScript has wrapped around the Java object.

Warnings:

- Beware that you don't confuse this JavaScript `getClass()` method with the Java `getClass()` method. Although they have the same name, they yield a different result. The JavaScript method yields a JavaScript object of the class JavaClass. The Java method yields a Java object of the class `java.lang.Class`, which is not the same thing.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Instantiate a Java object
myJavaString = new java.lang.String;
// Display its Java class
document.write("Java class : ");
document.write(myJavaString.getClass());
document.write("<BR><BR>");
// Display its JavaScript class
document.write("JavaScript class : ");
document.write(myJavaString.constructor.name);
document.write("<BR><BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	LiveConnect

JavaPackage object (Object/Navigator)

A JavaScript data type that encapsulates a Java Package.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	`myJavaPackage = myWindow.Packages`

A collection of Java classes are represented as a package. The complete set of all available Java Packages are contained within a single parent `JavaPackage` object. This constructs a hierarchy of `JavaPackage` and `JavaClass` objects.

The properties of the `JavaPackage` object are those that refer to any Java Class or Java Packages that belong to it. However, these are generally not enumerable so you do need to know what they are called before you can access them.

Because you cannot enumerate the properties of this object you won't be able to walk a Java package hierarchy with a script driven tree walker.

The `JavaPackage` tree structure resembles the directory structure that the Java class files are stored in. Some class files are collected into ZIP archives but even then the hierarchy is still intact and there will be one package that represents the ZIP file and corresponds to that node in the tree. Any arbitrary collection of classes will be represented by a `JavaPackage` object, regardless of how they are stored in the file system.

Example code:

```
// Create a JavaScript object that encapsulates a Java Package
var myJavaPackage = Packages.java.util;
```

See also: JavaScript to Java values, LiveConnect, `netscape.applet`, `Packages.java`, `Packages.netscape`, `Window.java`, `Window.Packages`, `Window.sun`

JavaScript Bookmark URLs (Advice)

Bookmarks composed of JavaScript calls.

You can build a bookmark by typing in a JavaScript URL. You do need to be careful as you type it in.

To create a JavaScript bookmark, open the bookmark manager and create a new entry. Be careful to only refer to objects that are likely to exist regardless of the page on view. You can crash the logic of the browser if your bookmark script tries to manipulate objects that belong to a page other than that which you are browsing.

It is necessary to work within these limitations because the user may request a bookmark at any time.

These JavaScript-coded bookmarks are sometimes known as Bookmarklets.

It is possible to create bookmarklets to aid in a number of information-gathering and debugging tasks. The example demonstrates how to display form elements in a pop up window. This only works if your bookmarks can hold a long enough URL value. It works fine in MSIE and Netscape 6.0, but not in version 4.0 of Netscape although you can paste it into the location bar and hit return for the same effect.

Upon testing it was found that only the Opera 5 browser has a security check to ask whether the script is allowed to check the password.

Example code:

```
// Provided for our enjoyment by Jon Stephens
// Note, the line breaks and space have been added to aid
// readability.  You should omit the line breaks when
// inserting this into the bookmark. Any unnecessary spaces
// can also be removed to prevent URL buffer overruns.
// This also works fine when wrapped in a function
// declaration and called from a button on the form page.
```

```
javascript:var output = "";

for(var i = 0; i < document.forms.length; i++)
{
   for(var j = 0; j < document.forms[i].elements.length; j++)
   {
      output += "Form " +  i+  " -- # " + j + " -- Type: "
        +  document.forms[i].elements[j].type +  "; "
        +  document.forms[i].elements[j].name +  ": "
        +  document.forms[i].elements[j].value +  "<br>";
   }
}
var newWin = window.open("","newWin","width=350,height=350");
with(newWin.document)
{
   open();
   write(output);
   close();
}
```

See also:	Bookmarklets, JavaScript interactive URL, javascript: URL

Cross-references:

Wrox *Instant JavaScript* –page 49

JavaScript debugger console (Advice)

Netscape JavaScript debugger console window for viewing error logs.

If you type the `javascript:` URL method into the location box on a Netscape 4 browser and then press return, the web browser will display a debugging console window that gives you some helpful messages about what is happening in your script. It can certainly speed up the debugging process when you are trying to trace a fault.

You can type short pieces of JavaScript source code into the type-in box and press return and the results will be executed in console.

For example, to see the value of some mathematical constants, you type these lines into the location box:

```
Math.PI

Math.LOG10

Number.MAX_VALUE
```

Warnings:

- A small bug in the console requires that the first item be triggered with two carriage returns. For some reason the console eats the first message and then spits it out with the second. Everything is fine after that.

See also:	Debugging –client side, JavaScript interactive URL, `javascript`: URL

Cross-references:

Wrox *Instant JavaScript* –page 48

JavaScript Document Source URL (Definition)

JavaScript can be called directly instead of fetching a page.

JavaScript URLs can be used in the `HREF` attribute of an anchor tag. If this technique is used, a fragment of JavaScript code can call in an external function and generate the content of another page without ever calling a web server for the page.

Quite sophisticated dynamic page generation can be accomplished like this, however, at some stage you will have to have downloaded the JavaScript code to be executed in the first place.

Example code:

```
<HTML>
<BODY>
<SCRIPT>
var the_value = 100;
function increment()
{
   the_value++;
}
</SCRIPT>
<A HREF="javascript:alert(the_value)">Click to display the value</A>
<A HREF="javascript:increment()">Click to increment the value</A>
</BODY>
</HTML>
```

See also:	JavaScript Image Source URL, `javascript`: URL

JavaScript embedded in Java (Definition)

Java applets can contain JavaScript code fragments that can be executed by the applet.

You can embed fragments of JavaScript inside Java applets although there are some limitations to what you can accomplish with this. For this to work, Java and JavaScript both have to be enabled and working. This may require some preference setting. Currently, it only works in Netscape.

The `<APPLET>` tag is the main way to embed a Java applet within an HTML document. That much generally works across all browsers.

JavaScript scripts should always have access to all the internals of a Java applet that it cares to make publicly available. However, the converse is not necessarily true. Java applets may not access the script-owned object space unless you give them permission to do so.

This is done with the `MAYSCRIPT` tag attribute on the `<APPLET>` tag.

The Java applet can then mimic JavaScript methods using the Java syntax, and there are mechanisms for accessing `eval()` functionality for interpreting JavaScript source text.

See also: Java, Java exception events, `JSObject` object, MAYSCRIPT

Cross-references:

Wrox *Instant JavaScript* –page 57

JavaScript entity (Pitfall)

This functionality is deprecated and should not be used in new projects.

The HTML character entities are useful for describing hard to type characters.

They are functionally similar to the back-quote substitutions that are available server-side with Netscape Enterprise Server.

MSIE version 3.0 introduced a means of passing JavaScript values into the HTML source space using a syntax that is similar to the character entity syntax. However, it is used in contexts that character entities were never intended to be used in. It is also advised as deprecated functionality in the HTML version 4.0 standard.

You might indicate an image width like this:

```
<IMG SRC="..." WIDTH="100">
```

The value for the image width can be taken from a JavaScript expression like this:

```
<IMG SRC="..." WIDTH="&{myWidth * myScaleFactor};">
```

This assumes that the values *`myWidth`* and *`myScaleFactor`* have already been defined in some earlier fragment of JavaScript.

The entity can be used to replace a single character in the tag attribute value so you can concatenate other characters such as percent signs if you use it in `<HR>` tags for example.

This functionality should be avoided and the usual client or server methods used to define the values in HTML tags.

Example code:

```
<HTML>
<BODY>
<FORM>
<INPUT TYPE="text" VALUE="&{top.name;}">
</FORM>
</BODY>
</HTML>
```

See also: Adding JavaScript to HTML, Backquote (`), Deprecated functionality, Pitfalls

Cross-references:

Wrox *Instant JavaScript* –page 47

JavaScript Image Source URL (Definition)

Image data can be generated in some browsers.

Within some limitations, you can generate image data within a web browser's JavaScript environment. This seems to work best on the Netscape version 4.0 browser, but it is worth exploring somewhat to see if it can be exploited on other browsers.

If you store this text into a script variable it describes a 1 pixel image:

```
#define x_width 1

#define x_height 1

static char x_bits[] = {0x00};
```

This is the code for an XBM image. It is only black and white but it may be useful to be able to create an image with a script.

Now you can call this as the source of an image like this:

```
<IMG SRC="javascript:variablename">
```

The example works as it is in Netscape 4.0. Adding a MIME type header will help browsers to determine what the content type is and this technique doesn't need to be confined to just image data.

Example code:

```
<HTML>
<BODY>
<SCRIPT>
var the_image_data;
the_image_data  = "Content-type: image/x-xbm";
the_image_data  = "";
the_image_data  = "#define x_width 10\n";
the_image_data += "#define x_height 1\n";
the_image_data += "static char x_bits[] = {";
the_image_data += "0xFF,0xFF,0xFF,0xFF,0xFF,";
the_image_data += "0xFF,0xFF,0xFF,0xFF,0xFF";
the_image_data += "};";
</SCRIPT>
<IMG SRC="javascript:the_image_data">
</BODY>
</HTML>
```

See also: JavaScript Document Source URL, javascript: URL, MIME types

Cross-references:

Wrox *Instant JavaScript* –page 49

JavaScript interactive URL (Request method)

An interactive JavaScript statement executor.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0

You can type short pieces of JavaScript source code into the location box and press return and the results will be executed in the document window.

For example, to see the value of some mathematical constants, you type these lines:

```
javascript:Math.PI

javascript:Math.LOG10

javascript:Number.MAX_VALUE

javascript:alert(top.location)
```

See also:	Bookmarklets, JavaScript Bookmark URLs, JavaScript debugger console, javascript: URL

Cross-references:

Wrox *Instant JavaScript* –page 48

JavaScript language (Overview)

JavaScript overview summary.

Availability:	ECMAScript edition –2

JavaScript is a so-called third generation language.

JavaScript was originally developed by Netscape Communications and is influenced by many other languages. Most notably, it borrows from:

- C language
- Pascal
- BASIC
- Java
- Perl
- Other scripting languages

This means that if you have already used any of these other languages, it is likely that you will become proficient in JavaScript fairly quickly.

JavaScript and Java are two distinctly different languages. They even operate in a different architectural way. JavaScript is interpreted while Java is compiled. It is a lot simpler to get started using JavaScript and designers find it's a good way to become accustomed to the more intense software developer issues.

Java code tends to be quite self-contained and not require the help of any additional systems. On the other hand, JavaScript is often more like a kind of glue that joins various parts of the web together so they can inter-operate more effectively.

JavaScript is object-based. That is to say, the basic language operation and access to the hosting environment is by means of objects that represent items in the real world. A JavaScript application is built from a collection of various kinds of objects, which communicate with one another.

Every JavaScript environment comes with a set of Core Objects. These provide the fundamental basis on which the language is built. It is these objects that are defined in the ECMAScript standard. There is a `Global` Object which is the topmost object in the hierarchy. Other objects such as the `Math` Object provide more specialized capabilities.

In JavaScript we talk about primitive values having a type. There are a set of standard types built-in to the core JavaScript interpreter. Although JavaScript is object-based, it is not a class based object oriented language, and so while these Primitive Types behave very much like object oriented classes, they aren't really implemented as such. The language does provide some facilities for creating new object types however.

An Object in the JavaScript environment is a collection of Properties which you can access by name. Some you can only access in a read-only fashion while others can be modified. The Properties have attributes such as a read-only flag, which controls this access to the values contained in the property. Properties may have other attributes as well and can often be a reference to another object. Requesting a property from an object can therefore yield another object, which in turn has properties of its own. If you think of the properties as containers you'll soon feel comfortable with the concept.

Objects can also respond to messages, each of which will invoke a named Method. Those methods may behave much like a property request or they may cause some action to happen. At the JavaScript programming level, methods and properties begin to lose their distinction. Methods are often analogous to Functions in other languages. Indeed, methods are implemented as functions that are attached to objects by means of properties. You may use functions in a traditional way, very much like you would in C, but the functions you use in the scripts are actually methods that belong to the `global` object. However, in that case you don't need to specify which object should execute the method.

The syntax of JavaScript intentionally mimics that of Java but many rules are relaxed to make the language easier to use. For example, you don't need to declare the type of a variable before using it. Properties don't need to have their type specified and functions do not need to be placed earlier in the script than when they are first invoked.

Although JavaScript is intended to be forgiving and therefore easy to be comfortable with and yet not be a hard-core programmer, beyond the simple basics, it certainly is not a simple language to learn and exploit fully.

See also:	Constant expression, Declaration, ECMAScript, Expression, Free-format language, History, int, JavaScript version, Lexical element, LiveScript, Native feature, Overview, Statement

Cross-references:

ECMA 262 edition 2 –section –4.2

ECMA 262 edition 3 –section –4.2

O'Reilly *JavaScript Definitive Guide* –page 2

Wrox *Instant JavaScript* –page 1

Wrox *Instant JavaScript* –page 11

JavaScript Style Sheets (Definition)

A standard for describing style sheets.

The JavaScript Style Sheet facility (JSS) is based on the CSS level 1 capabilities. They are only supported in the version 4.0 Netscape browsers and are now completely deprecated. Netscape 6.0 does not support them at all. This gives rise to some concerns on portability issues.

However, if your web server is capable of recognizing the user agent value and serving different content accordingly, you may be able to ensure that JSS style sheets are served only to requesting Netscape 4.0 browsers and that other browsers will be presented with CSS style sheets instead.

The main difference is in the notation used to describe the JSS rules.

There are two ways to call in a JSS defined style sheet, either with the `<STYLE>` tag, which embeds the style definitions into the page, or with the `<LINK>` tag, which includes them from an external document.

Creating styles in JavaScript comes down to correctly forming the object model that refers to the `style` properties of the `document` object. This is managed by means of a `tags` object, which is the root of a collection of objects for which the style can be defined as properties. By and large the `tags` object has a property that corresponds to every style controllable HTML tag. It's fairly easy to guess the likely syntax and give a CSS reference, you can probably figure out the properties of the individual objects. The properties correspond with those attributes of the instances of the tags described in the browser object model (or the DOM if you prefer).

There are some special methods and functions defined for JSS to be used when creating styles. These are:

- `margins()`
- `rgb()`
- `contextual()`

For JavaScript to be able to modify style settings, it needs to be able to read the current values and to be able to write new values into the object properties.

In Netscape 4.0, the absolute and relative style properties apply to positioning of an object. If these are present, the objects are assumed to be in layers. Layers can be controlled very effectively from within JavaScript. they can be moved, sized and have their visibility changed. However there are rendering issues and the effects are complex and hard to control when some styled objects are already rendered on the page. Some tags can have absolute positioning applied directly but it is not portable across minor revision changes of the browser.

MSIE version 4.0 provides dynamic style control. It is improved significantly in version 5.0 and 5.5 and the same DOM based model is built into Netscape 6.0. The styles are available as `host` objects. You can modify the appearance of styled object in MSIE even after they have already been displayed. Changing the style or position of an object gives you all the capabilities that layers provided in Netscape, and an automatic damage repair redraw is also triggered if necessary when the style of an object is modified.

Warnings:

- JSS comes about because internally Netscape 4.0 converts CSS rules into JavaScript code and executes it as such. That is why you lose styling when you turn off JavaScript for security reasons.
- Netscape decided to make this interface available to the developer and exposed it as JSS. This means there is a conflicting alternative to CSS. It is very unlikely that JSS would be supported in any other browser than Netscape 4.0 given that most browser manufacturers are striving for standards compliance. They will all be supporting CSS2 fully in due course.
- It is therefore recommended that while you may want to experiment with JSS, it is a deprecated item. As such you probably should avoid deploying it in any mission critical projects or using it in new developments.
- MSIE 3.0 does not allow JavaScript access to style properties.
- The JSS functionality is removed from Netscape 6.0.

Example code:

```
<STYLE TYPE="text/JavaScript">tags.P.borderWidth = 10;tags.B.color =
"#FF0000";</STYLE>
```

See also: `<META>`, `<STYLE TYPE="...">`, `<STYLE>`, Case Sensitivity, `contextual()`, CSS level 1, `Document object`, JSS, `JSSTag.apply`, `JSSTag.margins()`, `JSSTag.rgb()`, `JSSTags object`

Cross-references:

Wrox *Instant JavaScript* –page 50

JavaScript to Java values (Definition)

Conversion of JavaScript primitives and objects to Java compatible types.

When a Java method is called from JavaScript, the JavaScript values need to be converted to compatible and useful data types.

In some cases a single JavaScript type will be compatible with arguments of many different types. An example of this is the JavaScript Number primitive that will convert readily into half a dozen different Java types.

Some JavaScript data types correspond exactly to a single and specific Java type. An example of this is `null`.

A third possibility is that many different JavaScript types will become a single Java type. This is what happens to objects which all become a generic JavaScript object represented by the Java class `JSObject`.

Here is a table that summarizes the correspondence between JavaScript and Java types when converting to Java. This would be used when passing values into methods under control of JavaScript:

JavaScript	Java
boolean primitive	boolean
boolean primitive	java.lang.Boolean
null	null
number primitive	byte
number primitive	char
number primitive	double
number primitive	float
number primitive	int
number primitive	java.lang.Double
number primitive	long
number primitive	short
string primitive	java.lang.String
Function object	netscape.javascript.JSObject
JavaArray object	netscape.javascript.JSObject
JavaClass object	netscape.javascript.JSObject
JavaMethod object	netscape.javascript.JSObject
JavaObject object	The encapsulated Java object unwrapped
JavaPackage object	netscape.javascript.JSObject
netscape.javascript.JSObject	java.lang.String
All other JavaScript objects	netscape.javascript.JSObject

When this happens under the control of Java, there are further limitations as to what can be exchanged between the environments. This is discussed under the individual methods that are affected in the JSObject description.

This table summarizes the relationships at the passing interface. JavaScript values may have been converted during the expression evaluation. You may also need to coerce some values as they are passed so they are correctly mapped to the Java interface for the method.

Java objects are wrapped so that JavaScript can access them as a `JavaObject` object. When they are passed back to Java, the encapsulation is removed and the intrinsic Java object is accessed directly.

Warnings:

- Be aware that you can lose precision when passing data into and out of a Java applet method. As well as type conversion, value truncation may occur.
- JavaScript number values can interface to `java.lang.Double` method arguments but not to `java.lang.Integer` or `java.lang.Float`.

- Because JavaScript does not have a character data type, numbers can map to Java `char` data types. A character in a string in JavaScript is presented as a `String` primitive but is only one character long. This will convert to `java.lang.String` but will not naturally convert to a Java `char` data value unless you make some effort in the script to prepare it first. A `String.charCodeAt()` conversion may be necessary.
- Some Java encapsulations are not unwrapped when the value is passed to Java. A `JavaPackage`, `JavaArray`, `JavaClass` and the deprecated `JavaMethod` are all encapsulated as a `JSObject`. This may cause problems if the objects are passed backwards and forwards as the encapsulation might become increasingly nested. This behavior would be classed as erroneous and may be platform dependent. It may also depend on what happens inside the applet code when the method is called.
- JavaScript arrays are not converted to Java arrays of `JSObject` items. They are passed as a JSObject encapsulated as a whole.

See also:	Boolean, `Function` object, Java method data conversion, Java to JavaScript values, `JSObject` object, `JSObject.call()`, `JSObject.eval()`, `JSObject.getMember()`, JSObject.getSlot(), `JSObject.toString()`, LiveConnect, `netscape.javascript.JSObject, null`

JavaScript version (Standard)

The version history for JavaScript.

JavaScript was initially developed by Netscape Communications and was originally called LiveScript 1.0. Around that time, the Java language was becoming more popular and possibly as a marketing ploy, the name of LiveScript was changed to JavaScript. It was first available to the public in version 2.0 of Netscape.

Version	ECMA	Notes
LiveScript 1.0	No	The original precursor to JavaScript
JavaScript 1.0	No	Netscape 2.0 implemented this. Now mostly obsolete.
JavaScript 1.1	Yes	Supported in Netscape 3.0 and Netscape Enterprise Server 2.0. Also supported in Opera 3.0. More robust and better support for arrays. Image replacement and access to plugin properties. Scroll control.
JavaScript 1.2	No	Netscape 4.0 to 4.05 added `RegExp`, `switch` and `delete`. `Screen` object and interval timer. `Window move` & `resize`. Object and array literals added.
JavaScript 1.3	Yes	Version 4.06 to 4.76 of Netscape adds better exception handling. Also supported by Netscape Enterprise Server 3.
JavaScript 1.4	Yes	Netscape version 5.0 (not widely released).
JavaScript 1.5	Yes	Netscape 6.0 (final beta stages at time of writing).

See also:	History, JavaScript language, JScript version, LiveScript

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 3

Wrox *Instant JavaScript* –page 3

javascript: URL (Request method)

Execute some JavaScript code instead of fetching a document.

When you specify a URL in a web browser, the intent is usually to fetch a document from a remote web server.

The `javascript`: URL method is used to execute a fragment of JavaScript code when the URL is requested.

You can use the `javascript` URL as follows:

- To call up the debugger console
- To interactively execute statements
- As document source
- As bookmarks

The view-source: URL can be used in Netscape to call up a source view of a document under script control. Its not very portable and not much use for anything other than debugging.

These are all described in separate topics.

You can call up the JavaScript debugger by setting a document location to "`javascript`:", "`livescript`:" or "`mocha`:".

Looking at the internals of the Netscape browser, this debugging console is itself written in HTML with JavaScript dynamic actions.

Mostly, these special URLs will be useful for debugging –getting details of the disk cache may be useful for example. Pulling up the JavaScript debugger page if you detect an error in your script might also be a cool trick.

With a `javascript:` URL, you can also type the code directly into the location bar of your Netscape browser to see the results of evaluating it right away.

As of JavaScript version 1.1, you can use the void operator to discard the result of an expression.

This `javascript:` URL form is vailable in the WebTV set top boxes effective from the Summer 2000 release. However, it cannot be typed in manually by the user as it can be in the desktop computer based web browsers.

Warnings:

- This technique does not work with MSIE 3.0.
- The JavaScript debugger is not present in MSIE at all, although it may be possible to use the Visual J++ debugging tools if you have them installed.
- Almost too late for inclusion was a report that `History.back()` calls that worked in JScript 5.1 started to fail on upgrade to JScript 5.5 service pack 1. In the end it turned out to be related to calling a `javascript:` URL within an `<A HREF="...">` context. In earlier versions of MSIE, you could omit the single and double quotes around the URL. Version 5.5 is no longer forgiving that omission. This may affect other kinds of URL values and other HTML tag attributes in an MSIE 5.5 browser.

Example code:

```
<HTML>
<HEAD>
<SCRIPT>
function test()
{
alert("Test function called");
}
</SCRIPT>
</HEAD>
<BODY>
<DIV onClick="javascript:test();">Click on me</DIV>
</BODY>
</HTML>
```

See also:	Adding JavaScript to HTML, Bookmarklets, JavaScript Bookmark URLs, JavaScript debugger console, JavaScript Document Source URL, JavaScript Image Source URL, JavaScript interactive URL, mailbox: URL, mailto: URL, URL, void

JellyScript (Definition)

The JavaScript interpreter inside a WebTV set-top box is referred to as JellyScript.

This interpreter is based on the normal web browser JavaScript with a few limitations and some additional functionality. The interpreter functionality is mainly limited due to the small amount of memory available in the set-top box. Low memory is important to keep the manufacturing costs as small as possible.

The JellyScript interpreter underwent an upgrade in late Spring 2000 and was released for public use during the Summer. It is generally referred to as the Summer 2000 release.

Versions prior to Summer 2000 did not care about case-sensitivity of built-in property and method names. This was also the case with MSIE 3.0

The Summer 2000 version introduces support for user-defined properties and such like. This means that DHTML effects created by authoring tools such as DreamWeaver should work better in WebTV boxes that are shipped after this date or are upgraded in the field.

Event handling support is improved in the latest release. Earlier versions had problems with event handlers that had not yet completed their execution after 10 seconds. The WebTV documentation still suggests that the `Window.onunload` event handler is not used.

The `javascript:` URL format can be used with the minor limitation that it cannot be typed manually by the user.

Anonymous functions are properly supported by the WebTV set-top box from the Summer 2000 release onwards. Earlier versions of this product only partially supported anonymous functions.

Referring to page element objects by their names is more flexible. Versions of WebTV prior to the Summer 2000 release required that the references were fully qualified. This is still recommended but the JellyScript interpreter is now more forgiving, along the lines of the JScript interpreter in the MSIE browser. Some care still needs to be taken when building web pages for use in cross browser situations.

Security was enhanced in the Summer 2000 release. This is now correctly implemented inline with the rules of access to window content from different domains.

The `screen` property was added to the `Window` object and contains a reference to the `Screen` object that describes the size and attributes of the TV display screen.

The `Window.open()` method will behave differently in JellyScript when compared with the behavior in a normal computer-based web browser. It creates a pseudo window in an IFRAME and appends it to the end of the current display. The user can then scroll down to this window. On the whole this behavior is so different to the normal `Window.open()` usage that it is probably best to avoid using it.

The Summer 2000 release of JellyScript now supports full double-precision math routines. Date object support is now up to ECMA standard specifications with full support for all properties and methods. The identity (`===`) and non identity (`!==`) operators are also now supported (consistent with the support in the MSIE browser).

There are some areas of functionality that are considered weak or are completely unsupported on this platform:

- `Window.open()`
- Regular Expression support
- Signed scripts
- Custom sort ordering
- Array sizes limited to 32768 elements
- Error handling is silent

Be careful when accessing form elements within a document. In fact you should generally fully qualify all references to objects within the page. This is mandatory with form elements, which should have the `document` object referred to as the first item in the hierarchy chain that describes the element's location in the DOM.

You should also be careful when testing for browser types in the user agent string. Eliminate the possibility of the browser being a WebTV box before testing for the Netscape browser. If you don't, then it is likely you will test `true` for being a Netscape browser even when executing the JavaScript code on a WebTV box. You will need to scan the entire user agent string. To parse out the string "WebTV" it might be worth converting to lowercase first to increase your chances of a match.

See also: TV Set-top boxes, WebTV

Web-references:

This link leads to a useful WebTV JavaScript guide, for interested parties.

http://developer.webtv.net/authoring/javascript/javascript.htm

.js (File extension)

JavaScript include file.

You can include external shared fragments of JavaScript into a web page by calling in a `.js` file.

A `.js` file simply contains that JavaScript which would have been placed inside a `<SCRIPT>` tag.

In the case of the Netscape browser, it can store configuration and preferences data in `.js` files.

These `.js` files can also be used server-side with Netscape Enterprise Server. They can be compiled with the LiveWire JavaScript compiler and linked with the HTML to create `.web` files that Netscape Enterprise Server can deliver very quickly to a requesting client browser.

The MIME type for a `.js` file used to be `application/x-javascript` but is now `text/javascript`. Either should work but `text/javascript` is preferred. This may necessitate you carrying out some server configuration changes if you don't already serve this kind of file.

The file is included by specifying its URL with the `SRC="..."` HTML tag attribute. You must also include a closing `</SCRIPT>` tag because `<SCRIPT>` is a block level item.

Example code:

```
<SCRIPT SRC="http://www.mydomain.com/include.js"></SCRIPT>
```

See also: `<SCRIPT ARCHIVE="...">`, `<SCRIPT SRC="...">`, File extensions, Preferences, Source files, Web browser

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 215

Wrox *Instant JavaScript* –page 3

.jsc (File extension)

JavaScript configuration file.

See also: netscape.lck, Preferences

JScript version (Standard)

The version history for JScript.

Microsoft added scripting capabilities to version 3.0 of the MSIE browser. The JScript interpreter is perhaps named differently so that detractors would not accuse Microsoft of modifying the JavaScript language.

Version 1.0 of JScript was also available in Internet Information Server as well as the MSIE version 3.0 browser. That first version of JScript is broadly compatible with JavaScript version 1.0 although there are differences between them:

Interpreter	ECMA	Notes
JScript 1.0	No	Equivalent to JavaScript 1.0 and released with MSIE 3.0
JScript 1.1	No	Never released
JScript 1.2	No	Evidence of its existence but status unknown
JScript 2.0	No	Released with IIS 1.0
JScript 3.0	Yes	Equivalent to JavaScript 1.2 and released with MSIE 4.0, IIS 4.0 and WSH 1.0 (some features are in MSIE 3.02)
JScript 4.0	Yes	Released with Visual Studio 6.0
JScript 5.0	Yes	Equivalent to JavaScript 1.5 and supported on all 32 bit Windows operatings systems with MSIE.
JScript 5.1	Yes	Released with IIS 5.0 on Windows 2000
JScript 5.5	Yes	Released with MSIE 5.5

The JScript support in MSIE is excellent. However, if you limit yourself purely to the JavaScript sub-set, there are some limitations in the support. The language has been extended somewhat but not in the same way as Netscape Communications Inc provided enhancements to their interpreter.

If the JScript interpreter is reinstalled without upgrading the browser, you may be using a version of JScript that is later than the browser. Scripts should work normally as long they do not exploit features of Jscript that have changed. However, the scripts may not be able to access some new features of the later version. For example, JScript 5.5 can be installed over the top of JScript 5.0 in a version 5.0 MSIE browser. You get the later core language features but continue to use the old document model. This can get utterly confusing for script developers.

See also:	History, Internet Explorer, JavaScript version

Cross-references:

Wrox *Instant JavaScript* –page 3

.jse (File extension)

Nombas ScriptEase source file.

See also:	File extensions, Nombas ScriptEase, Standalone JavaScript

.jsh (File extension)

Nombas ScriptEase include file.

See also:	File extensions, Nombas ScriptEase, Standalone JavaScript

JSObject object (Java class)

A Java class that encapsulates JavaScript objects for access from Java code.

Availability:	JavaScript –1.1 Netscape –3.0	
JavaScript syntax:	N	*myJSObject* = netscape.javascript.JSObject
Class methods:	getWindow()	
Object methods:	call(), eval(), getMember(), getSlot(), removeMember(), setMember(), setSlot(), toString()	

This Java class is otherwise known as `netscape.javascript.JSObject` (to give it its full name within the Java context). This provides a way for Java code to interact fully with the JavaScript native environment.

`JSObject` is a sub-class of the generic `Object` class within Java. Its public interface defines the following methods and properties:

- `getWindow()`
- `getMember()`
- `getSlot()`
- `setMember()`
- `setSlot()`
- `removeMember()`
- `call()`
- `eval()`
- `toString()`

Note that these are hooks to Java methods although they may look like JavaScript methods.

When member properties and slot values are accessed from arrays, you get an `Object` object returned. If this object is really a JavaObject, then it will be unwrapped and the encapsulated Java object will be returned without its JavaScript wrapper. It will still be returned as an `Object` object but it can then be cast to a native Java object type rather than another JSObject.

The `setMember()` and `setSlot()` methods perform the converse although there are some subtle limitations.

Your Java development environment should give you plenty of help with the compilation of applets. The key point is that you have a copy of the `netscape.javascript.JSObject` class available for the applet to be linked against. This may involve setting your CLASSPATH to defined where the Java classes are located. The file you need may be browser version specific. In Netscape Navigator version 4.0, the file is called `java40.jar` but it may be named differently in other versions. Where it is located also may depend on how and where you installed Netscape.

See also:	Java calling JavaScript, Java to JavaScript values, JavaScript embedded in Java, JavaScript to Java values, `JSObject.call()`, `JSObject.eval()`, `JSObject.getMember()`, `JSObject.getSlot()`,`JSObject.getWindow()`, `JSObject.removeMember()`, `JSObject.setMember()`, `JSObject.setSlot()`, `JSObject.toString()`, LiveConnect, MAYSCRIPT, `netscape.javascript.JSObject`

Method	JavaScript	JScript	N	IE	Opera	Notes
`call()`	1.1 +	-	3.0 +	-	-	-
`eval()`	1.1 +	-	3.0 +	-	-	-
`getMember()`	1.1 +	-	3.0 +	-	-	-
`getSlot()`	1.1 +	-	3.0 +	-	-	-
`removeMember()`	1.1 +	-	3.0 +	-	-	-
`setMember()`	1.1 +	-	3.0 +	-	-	-
`setSlot()`	1.1 +	-	3.0 +	-	-	-
`toString()`	1.1 +	-	3.0 +	-	-	-

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page 8-570

Wrox *Professional JavaScript* –page 544

JSObject.call() (Java method)

Calls a method in the JavaScript object from the Java environment.

Availability:	JavaScript –1.1 Netscape –3.0
Property/method value type:	`Object object`

Java syntax:	*myJSObject*.call("*aMethod*", *anArgArray*)	
Argument list:	*aMethod*	The name of a method to call.
	anArgArray	An array of arguments to pass to the method

This is the way in which a Java applet can call back to a JavaScript function in a page. Once you know the window, you can invoke methods that belong to it as well as access properties. This will always yield an `Object` object as a result.

There are quite restricted Java to JavaScript limitations on passing non-primitive values in the arguments array.

The values passed to JavaScript will conform to the following conversions as they are passed to the JSObject methods:

Java	JavaScript
java.lang.Boolean	JavaObject object
java.lang.Double	JavaObject object
java.lang.Integer	JavaObject object
java.lang.String	JavaObject object
netscape.javascript.JSObject	generic JavaScript object
all other Java objects	JavaObject object

The return values will conform to the following conversions as they are passed between the environments:

JavaScript	Java
boolean primitive	java.lang.Boolean
number primitive	java.lang.Double
string primitive	java.lang.String
JavaObject object	The encapsulated Java object unwrapped
all other JavaScript objects	netscape.javascript.JSObject

The result of this method call will be an `Object` object which needs to be cast to some other value for use in the Java environment.

See also:	Call, Java to JavaScript values, JavaScript to Java values, `JSObject` object, `JSObject`.eval(), LiveConnect

Cross-references:

Wrox *Professional JavaScript* –page 542-3

JSObject.eval() (Java method)

A means of invoking native JavaScript `eval()` functionality.

Availability:	JavaScript –1.1 Netscape –3.0	
Java syntax:	*myJSObject*`.eval("`*someScript*`")`	
Argument list:	*someScript*	Some valid JavaScript source

This is a much simpler way to execute JavaScript than by the `call()` method. Here there is no need to construct an array to pass in the method arguments.

The return values will conform to the following conversions as they are passed between the environments:

JavaScript	Java
boolean primitive	java.lang.Boolean
number primitive	java.lang.Double
string primitive	java.lang.String
JavaObject object	The encapsulated Java object unwrapped
all other JavaScript objects	netscape.javascript.JSObject

The `JSObject.eval()` method eliminates many of the parameter passing problems associated with the `JSObject.call()` method as far as type conversion is concerned. You will need to convert any parameters you want to pass into strings, but this does allow you to pass primitive values which you simply cannot do with the `JSObject.call()` method.

See also:	Eval code, eval(), JavaScript to Java values, JSObject object, JSObject.call(), LiveConnect

JSObject.getMember() (Java method)

Returns the value of a named property of the object belonging to a JavaScript object to a calling in the Java environment.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`Object object`	
Java syntax:	*myJSObject*`.getMember("`*aMemberName*`")`	
Argument list:	*aMemberName*	The name of a property belonging to the JSObject

As you read properties of JSObjects, you get more JSObjects returned. In this way, you can walk the document hierarchy to locate any item in the window referred to by the root JSObject.

The return values will conform to the following conversions as they are passed between the environments:

JavaScript	Java
boolean primitive	java.lang.Boolean
number primitive	java.lang.Double
string primitive	java.lang.String
JavaObject object	The encapsulated Java object unwrapped
all other JavaScript objects	netscape.javascript.JSObject

The result of this method call will be an `Object` object which needs to be cast to some other value for use in the Java environment.

See also:	JavaScript to Java values, `JSObject` object, LiveConnect

JSObject.getSlot() (Java method)

A means of accessing elements within an array encapsulated in a JSObject.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`Object object`	
Java syntax:	*`myJSObject`*`.getSlot(`*`anIndex`*`)`	
Argument list:	*`anIndex`*	The array index equivalent to [*`anIndex`*]

This is a means of accessing array elements in JavaScript arrays when they are encapsulated inside a JSObject.

The return values will conform to the following conversions as they are passed between the environments:

JavaScript	Java
boolean primitive	java.lang.Boolean
number primitive	java.lang.Double
string primitive	java.lang.String
JavaObject object	The encapsulated Java object unwrapped
all other JavaScript objects	netscape.javascript.JSObject

The result of this method call will be the element of the array at the slot location returned as an `Object` object which needs to be cast to some other value.

See also:	JavaScript to Java values, `JSObject` object, LiveConnect

JSObject.getWindow() (Java static method)

A static method to return a new JSObject that belongs to the window containing the applet.

Availability:	JavaScript –1.1 Netscape –3.0	
Property/method value type:	`JSObject object`	
JavaScript syntax:	N	`new JSObject(`*`anApplet`*`)`
Java syntax:	*`myJSObject`*`.getWindow(`*`anApplet`*`)`	
Argument list:	*`anApplet`*	The applet whose window is to be referenced by the new JSObject

When called from a Java applet, this method returns a new JSObject for the window containing the applet.

This is a way of creating a JSObject that relates to the correct window, that is, the one containing the applet. This factory method is called because there is no constructor for the JSObject class. It creates a JSObject appropriate for the applet whose reference is passed in its only parameter.

Example code:

```
// Create a JSObject for the applet we are running in
JSObject myJSObject = JSObject.getWindow(this);
```

See also:	`JSObject` object, LiveConnect

JSObject.removeMember() (Java method)

Remove a property from a JavaScript object.

Availability:	JavaScript –1.1 Netscape –3.0
Java syntax:	*`myJSObject`*`.removeMember()`

This is equivalent to the delete property mechanism in JavaScript.

See also:	delete, `JSObject` object, LiveConnect

JSObject.setMember() (Java method)

Stores a new value in a property.

Availability:	JavaScript –1.1 Netscape –3.0	
Java syntax:	`myJSObject.setMember("aName", "aValue")`	
Argument list:	*aName*	The name of the property to be changed
	aValue	The new value to be stored in the property

This method allows the Java code to set a property of a JSObject to a new value There is a minor limitation in that you must pass a Java object and cannot set a primitive value.

The values passed to JavaScript will conform to the following conversions as they are passed to the JSObject methods:

Java	JavaScript
java.lang.Boolean	JavaObject object
java.lang.Double	JavaObject object
java.lang.Integer	JavaObject object
java.lang.String	JavaObject object
netscape.javascript.JSObject	generic JavaScript object
all other Java objects	JavaObject object

See also:	Java to JavaScript values, `JSObject` object, LiveConnect

JSObject.setSlot() (Java method)

Store an element in the JavaScript array.

Availability:	JavaScript –1.1 Netscape –3.0	
Java syntax:	`myJSObject.setSlot(anIndex, "aValue")`	
Argument list:	*anIndex*	The array index equivalent to [*anIndex*].
	aValue	The new value to be stored in the array element

This method allows the Java code to set an element of a JavaScript array stored in a `JSObject`. There is a minor limitation in that you must pass a Java object and cannot set a primitive value.

The values passed to JavaScript will conform to the following conversions as they are passed to the JSObject methods:

Java	JavaScript
java.lang.Boolean	JavaObject object
java.lang.Double	JavaObject object
java.lang.Integer	JavaObject object
java.lang.String	JavaObject object
netscape.javascript.JSObject	generic JavaScript object
all other Java objects	JavaObject object

See also:	Java to JavaScript values, JSObject object, LiveConnect

JSObject.toString() (Java method)

Converts the object to a string value.

Availability:	JavaScript –1.1 Netscape –3.0
Java syntax:	`myJSObject.toString()`

The string equivalent value of the object is returned as a Java String.

See also:	JavaScript to Java values, `JSObject` object, LiveConnect, `ToString`, Type conversion

JSS (Definition)

A standard for describing style sheets in Netscape 4.0.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	JavaScript Style Sheets

JSSClasses object (Object/JSS)

A collection of JavaScript Style Sheet classes.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myJSSClasses* = *myDocument*.classes
Object properties:	className	

This style sheet control mechanism is becoming deprecated as it is only supported on Netscape 4.0 and will not be ratified by a W3C standard. It is not recommended that you use these facilities in new projects.

This object is somewhat like an array in that it contains a collection of objects that can be accessed associatively by name. However, unlike an array, it does not respond to the length property request. Also unlike an array, you cannot access its members using index values.

The only meaningful property of this object is one of its array elements corresponding to a named class in the style sheet. That property is also associated with a named CLASS="..." attribute of an HTML tag in the document.

You cannot enumerate this object to inspect its properties.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Document.classes[]

Property	JavaScript	JScript	N	IE	Opera	Notes
className	1.2 +	-	4.0 +	-	-	WarningDeprecated

Property attributes:

DontEnum.

JSSClasses.className (Property)

A JSS object corresponding to a single style class.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`JSSTags object`	
JavaScript syntax:	N	*myDocument*.classes.*aClassName*
Argument list:	*aClassName*	A named class within the style sheet

This object represents a class within the style sheet. However it is only available in Netscape 4.0 and is a means of access that leads ultimately to a `Style` object.

This mechanism is radically different to and much more complex than the simple `style` property belonging to an `Element` object in MSIE and Netscape 6.0.

From the value in this property, you would traverse the style tree down another level to find a style collection relating to a particular tag.

You cannot enumerate this object to inspect its properties.

This was mainly provided to give access to a style definition in Netscape 4.0 as opposed to the way it is used in MSIE and Netscape 6.0, where it is used to apply a style definition to a styled element within the document.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

JSSTag object (Object/JSS)

A single style object for use in Netscape 4.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myJSSTag* = *myDocument*.classes.*aClassName*.*aTagName*
	N	*myJSSTag* = *myDocument*.contextual(...)
	N	*myJSSTag* = *myDocument*.ids.*anElementName*
	N	*myJSSTag* = *myDocument*.tags.*aTagName*
	N	*myJSSTag* = *myJSSTags*.*aTagName*

Argument list:	*aClassName*	A named class within the style sheet
	anElementName	The value of a NAME="..." or ID="..." tag attribute.
	aTagName	The name of an HTML tag
Object properties:	align, apply, background, backgroundColor, backgroundImage, bgColor, borderBottomWidth, borderColor, borderLeftWidth, borderRightWidth, borderStyle, borderTopWidth, clear, clip, color, display, fontFamily, fontSize, fontStyle, fontWeight, height, left, lineHeight, listStyleType, marginBottom, marginLeft, marginRight, marginTop, paddingBottom, paddingLeft, paddingRight, paddingTop, textAlign, textDecoration, textIndent, textTransform, top, verticalAlign, visibility, whiteSpace, width, zIndex	
Object methods:	borderWidths(), margins(), paddings(), rgb()	

This is the Netscape 4.0 JSS equivalent of the DOM style object. You assign values to the properties of this object to define the styles according to the JSS rules. Browsers sometimes use different object types with incompatible properties and methods to represent the same thing. We cover them as distinctly different objects where it seems sensible. The Netscape 4.0 style settings are properties of a JSSTag object. Refer to the style object for details of the MSIE and Netscape 6.0 style control properties.

The property values for this object each represent a style attribute of an HTML tag.

To define a style setting with JSS, assign a value to this property according to the class name and tag name hierarchy.

These values are write-only and must be defined in the <HEAD> of the document. You cannot read them back or change them after the <BODY> has commenced loading.

You cannot enumerate the properties of this object so it is impossible to inspect them. Indeed, after repeated attempts to access them, they appear to be write-only properties.

Because you can only define them during the <HEAD> of a document, they don't provide much helpful facilities as regards dynamic style control.

It is highly recommended that you refrain from using these JSS facilities in any new projects. They are deprecated now that Netscape 6.0 adopts a more standardized DOM based approach to style settings.

The CSS support in Netscape 4.0 is available up to CSS level 1. In MSIE and Netscape 6.0, much of CSS level 2 is available through its more sophisticated and easier to manage style model.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Document.contextual(),JSSTags object, style object (2)

Property	JavaScript	JScript	N	IE	Opera	Notes
align	1.2 +	-	4.0 +	-	-	Warning, Deprecated
apply	1.2 +	-	4.0 +	-	-	Warning, Deprecated
background	1.2 +	-	4.0 +	-	-	Warning, DontEnum., Deprecated
backgroundColor	1.2 +	-	4.0 +	-	-	Warning, Deprecated
backgroundImage	1.2 +	-	4.0 +	-	-	Warning, Deprecated
bgColor	1.2 +	-	4.0 +	-	-	Warning, DontEnum., Deprecated
borderBottomWidth	1.2 +	-	4.0 +	-	-	Warning, Deprecated
borderColor	1.2 +	-	4.0 +	-	-	Warning, Deprecated
borderLeftWidth	1.2 +	-	4.0 +	-	-	Warning, Deprecated
borderRightWidth	1.2 +	-	4.0 +	-	-	Warning, Deprecated
borderStyle	1.2 +	-	4.0 +	-	-	Warning, Deprecated
borderTopWidth	1.2 +	-	4.0 +	-	-	Warning, Deprecated
clear	1.2 +	-	4.0 +	-	-	Warning, Deprecated
clip	1.2 +	-	4.0 +	-	-	Warning, DontEnum., Deprecated
color	1.2 +	-	4.0 +	-	-	Warning, Deprecated
display	1.2 +	-	4.0 +	-	-	Warning, Deprecated
fontFamily	1.2 +	-	4.0 +	-	-	Warning, Deprecated
fontSize	1.2 +	-	4.0 +	-	-	Warning, Deprecated
fontStyle	1.2 +	-	4.0 +	-	-	Warning, Deprecated
fontWeight	1.2 +	-	4.0 +	-	-	Warning, Deprecated
height	1.2 +	-	4.0 +	-	-	Warning, Deprecated
left	1.2 +	-	4.0 +	-	-	Warning, DontEnum., Deprecated
lineHeight	1.2 +	-	4.0 +	-	-	Warning, Deprecated
listStyleType	1.2 +	-	4.0 +	-	-	Warning, Deprecated
marginBottom	1.2 +	-	4.0 +	-	-	Warning, Deprecated
marginLeft	1.2 +	-	4.0 +	-	-	Warning, Deprecated
marginRight	1.2 +	-	4.0 +	-	-	Warning, Deprecated
marginTop	1.2 +	-	4.0 +	-	-	Warning, Deprecated
paddingBottom	1.2 +	-	4.0 +	-	-	Warning, Deprecated
paddingLeft	1.2 +	-	4.0 +	-	-	Warning, Deprecated
paddingRight	1.2 +	-	4.0 +	-	-	Warning, Deprecated
paddingTop	1.2 +	-	4.0 +	-	-	Warning, Deprecated
textAlign	1.2 +	-	4.0 +	-	-	Warning, Deprecated
textDecoration	1.2 +	-	4.0 +	-	-	Warning, Deprecated
textIndent	1.2 +	-	4.0 +	-	-	Warning, Deprecated

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	Notes
textTransform	1.2 +	-	4.0 +	-	-	Warning, Deprecated
top	1.2 +	-	4.0 +	-	-	Warning, DontEnum., Deprecated
verticalAlign	1.2 +	-	4.0 +	-	-	Warning, Deprecated
visibility	1.2 +	-	4.0 +	-	-	Warning, DontEnum., Deprecated
whiteSpace	1.2 +	-	4.0 +	-	-	Warning, Deprecated
width	1.2 +	-	4.0 +	-	-	Warning, Deprecated
zIndex	1.2 +	-	4.0 +	-	-	Warning, DontEnum., Deprecated

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
borderWidths()	1.2 +	-	4.0 +	-	-	-	-	-	1 +	-	Warning, Deprecated
margins()	1.2 +	-	4.0 +	-	-	-	-	-	1 +	-	Warning, Deprecated
paddings()	1.2 +	-	4.0 +	-	-	-	-	-	1 +	-	Warning, Deprecated
rgb()	1.2 +	-	4.0 +	-	-	-	-	-	-	-	Warning, Deprecated

Property attributes:

DontEnum.

JSSTag.align (Property)

Specified the alignment of objects.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myJSSTag*.align
	N	*myJSSTag*.align = *aValue*
CSS syntax:	float	
HTML syntax:	<... ALIGN="...">	
Argument list:	*aValue*	An alignment value

The `JSSTag.align` property does not correspond to any CSS property as there is not a specific `align` property in CSS even though some HTML 4.0 tags do.

It is based on values that are specified with HTML `ALIGN="..."` tag attributes.

The alignment of the styled object with respect to its containing parent object is defined in this property. The folllowing expected and widely available set of alignment specifiers are available:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

JSSTag.apply (Property)

A special JSS supporting property that holds a reference to a callback function.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Function object`	
JavaScript syntax:	N	*`myTags.anItem.apply = aHandler`*
Argument list:	*`aHandler`*	A JavaScript function to call when setting the object's style
	`anItem`	A tag name such as P or B or H1

This is a mechanism that allows the style engine in Netscape 4.0 to call for help when things get too complicated. Basically it calls the handler when an object is created and that handler sets the properties of the receiving object. There is an implied `this` on the front of any property names that are assigned with new values.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0

See also:	JavaScript Style Sheets

JSSTag.backgroundColor (Property)

The background color for objects.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated
Property/method value type:	`String primitive`
JavaScript syntax:	N *myJSSTag*.backgroundColor N *myJSSTag*.backgroundColor = "transparent" N *myJSSTag*.backgroundColor = *aColor*
CSS syntax:	background-color: *aColor*background-color: transparent
HTML syntax:	<BODY BGCOLOR="...">
Argument list:	*aColor* color value specified as #numeric or color name

The `JSSTag.backgroundColor` property corresponds to the `background-color` CSS property.

This property controls the background color of the element.

The value should be a color specified using the normal numeric or symbolic name notation.

In addition, the value "`transparent`" can be specified to allow the background to be "see-through".

The default value is "`transparent`".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`style.backgroundColor`

JSSTag.backgroundImage (Property)

The URL of a background image for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.backgroundImage
	N	*myJSSTag*.backgroundImage = "none"
	N	*myJSSTag*.backgroundImage = *aURL*
CSS syntax:	background-image: *aURL*	
HTML syntax:	<BODY BACKGROUND="...">	
Argument list:	*aURL*	A URL that points at an image

The JSSTag.backgroundImage property corresponds to the background-image CSS property.

This property is used to specify the URL of an image file to be loaded and used as the background. Alternatively the value "none" can be specified to deactivate the image background.

The default value for this property is "none".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Background.src, style.backgroundImage

JSSTag.borderBottomWidth (Property)

The thickness of the bottom edge of border round objects.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.borderBottomWidth`
	N	*myJSSTag*`.borderBottomWidth = `*aWidth*
CSS syntax:	`border-bottom-width: `*aWidth*	
HTML syntax:	`<... MARGINHEIGHT="...">`	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.borderBottomWidth` property corresponds to the `border-bottom-width` CSS property.

The content of an object is enclosed in a bounding extent rectangle. Around this is placed a padding space controlled by the padding parameters. The padding expands the bounding. Then the border is placed around this. The border width increases the bounding rectangle around the object still further. Last of all the margin is placed around the outside. This forms a rectangle that is used to locate the object adjacent to any other objects or page margins.

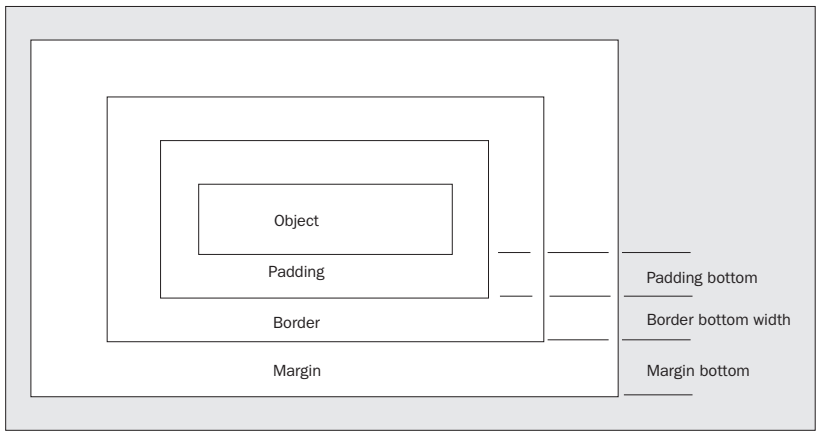
The value of this property can be specified as follows:

- thin
- medium
- thick

It can also be specified with a length value in one of several units of measure.

The initial value is set to "medium".

This value is not inherited from its parent container element.



Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.borderColor (Property)

The color of the border around objects.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.borderColor
	N	*myJSSTag*.borderColor = *aColor*
CSS syntax:	border-color: *aColor*	
HTML syntax:	<... BORDERCOLOR="...">	
Argument list:	*aColor*	color value specified as #numeric or color name

The JSSTag.borderColor property corresponds to the border-color CSS property.

This property controls the color of all four borders of the object. The value is a color specified in the normal numeric or symbolically named fashion.

The initial value is taken from the color property of the element around which the border is placed.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.borderColor

JSSTag.borderLeftWidth (Property)

The thickness of the left edge of the border around objects.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.borderLeftWidth
	N	*myJSSTag*.borderLeftWidth = *aWidth*
CSS syntax:	border-left-width: *aWidth*	
HTML syntax:	<... MARGINWIDTH="...">	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.borderLeftWidth` property corresponds to the `border-left-width` CSS property.

Refer to the `JSSTag.borderBottomWidth` topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.borderRightWidth (Property)

The thickness of the right edge of the border around objects.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.borderRightWidth
	N	*myJSSTag*.borderRightWidth = *aWidth*
CSS syntax:	border-right-width: *aWidth*	

HTML syntax:	<... MARGINWIDTH="...">	
Argument list:	*aWidth*	A CSS length value

The JSSTag.borderRightWidth property corresponds to the border-right-width CSS property.

Refer to the JSSTag.borderBottomWidth topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.borderStyle (Property)

The type of line used for the border around objects.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myJSSTag*.borderStyle
	N	*myJSSTag*.borderStyle = *aStyle*
CSS syntax:	border-style: *aStyle*	
Argument list:	*aStyle*	A border style value

The JSSTag.borderStyle property corresponds to the border-style CSS property.

The following style values can be applied to borders with this property:

- none
- hidden
- dotted
- dashed
- solid
- double
- groove
- ridge

- inset
- outset

The initial value for this property is "none".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.borderStyle

JSSTag.borderTopWidth (Property)

The thickness of the top edge of the border around objects.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.borderTopWidth
	N	*myJSSTag*.borderTopWidth = *aWidth*
CSS syntax:	border-top-width: *aWidth*	
HTML syntax:	<... MARGINHEIGHT="...">	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.borderTopWidth` property corresponds to the `border-top-width` CSS property.

Refer to the `JSSTag.borderBottomWidth` topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.borderWidths() (Method)

Sets all of the border width values for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myJSSTag*.borderWidths(*aTop*, *aHoriz*, *aBottom*)
	N	*myJSSTag*.borderWidths(*aTop*, *aRight*, *aBottom*, *aLeft*)
	N	*myJSSTag*.borderWidths(*aValue*)
	N	*myJSSTag*.borderWidths(*aVert*, *aHoriz*)
CSS syntax:	border-width: *aTop*, *aHoriz*, *aBottom*border-width: *aTop*, *aRight*, *aBottom*, *aLeft*border-width: *aValue*border-width: *aVert*, *aHoriz*border: *aValue*	
HTML syntax:	<... BORDER="...">	
Argument list:	*aBottom*	A border width applied at the bottom
	aHoriz	A border width applied left and right
	aLeft	A border width applied to the left
	aRight	A border width applied to the right
	aTop	A border width applied at the top
	aValue	A border width value applied all round
	aVert	A border width applied top and bottom
Event handlers:	This controls the border attribute of an image objects	

This method provides a way to set all four border width values at once. The number of arguments is variable and the values are applied in different ways according to how many are supplied.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.borderWidth

Event name	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
This controls the border attribute of an image object.	1.2 +	-	4.0 +	-	-	-	-	-	1 +	-	Warning, Deprecated

JSSTag.clear (Property)

A property that clears the style and forces the object to be displayed below a left or right aligned image.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.clear
	N	*myJSSTag*.clear = *aControl*
CSS syntax:	clear: *aControl*	
HTML syntax:	<... CLEAR="...">	
Argument list:	*aControl*	a clear control value

The `JSSTag.clear` property corresponds to the `clear` CSS property.

This property is used to control the floating elements around the element to which the styling applies. Clearing the float attribute to one side or the other forces a line break on that side and will not allow floating elements to flow into an adjacent position.

The following values are permitted:

- none
- left
- right
- both

The default value for this property is "none".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.clear

JSSTag.color (Property)

A foreground color for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape Navigator version –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.color`
	N	*myJSSTag*`.color = `*aColor*
CSS syntax:	`color: aColor`	
HTML syntax:	`<... COLOR="...">`	
Argument list:	*aColor*	color value specified as #numeric or color name

The `JSSTag.color` property corresponds to the color CSS property.

This property controls the foreground color of text in the element.

The value should be a color specified using the normal numeric or symbolic name notation.

The default value depends on the settings in the user preferences of the user agent (the browser).

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	color names, color value, `style.color`

JSSTag.display (Property)

Controls the display visibility of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.display`
	N	*myJSSTag*`.display = `*aControl*
CSS syntax:	`display: `*aControl*	
Argument list:	*aControl*	a display control value

The `JSSTag.display` property corresponds to the `display` CSS property.

This property controls the way that a styled element is displayed on-screen. It can cause the element to appear as a block item, something that is inline, a list or other possibilities. The following values are accepted:

- inline
- block
- list-item
- run-in
- compact
- marker
- table
- inline-table
- table-row-group
- table-header-group
- table-footer-group
- table-row
- table-column-group
- table-column
- table-cell
- table-caption
- none

Elements having the display property set to "block" will be forced to start on a new line.

The "inline" value keeps items running together on the same line.

Setting the display property to "none" will completely hide an item.

The other property values are quite complex and beyond the scope of our coverage here and are documented thoroughly in other works that specifically cover the CSS presentation styling standard. In any case, the whole JSS style complex is deprecated and will likely fall into disuse.

The default value for this property is "inline".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`style.display`

JSSTag.fontFamily (Property)

The font family or typeface name for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.fontFamily
	N	*myJSSTag*.fontFamily = *aFontList*
CSS syntax:	font-family: *aFontList*	
HTML syntax:	<FONT FACE="...">	
Argument list:	*aFontList*	A list of one or more font names

The `JSSTag.fontFamily` property corresponds to the `font-family` CSS property in a style sheet. It allows you to select a font for use in the object within the document.

Some example values of specific font families are listed below:

- Arial
- Courier
- Times
- Garamond
- Palatino
- Helvetica
- "New Century Schoolbook"

Font names containing spaces will need to be enclosed in quotes.

These are generic family names and allow the browser to choose its own best matching font:

- serif
- sans-serif
- monospace
- cursive
- fantasy

You should provide your list of fonts as a comma separated list with the most preferred fonts to the left.

The default value for this property depends on the browser font settings in the user preferences.

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.fontFamily

JSSTag.fontSize (Property)

The font size for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.fontSize
	N	*myJSSTag*.fontSize = *aSizeValue*
CSS syntax:	font-size: *aSizeValue*	
HTML syntax:	<FONT SIZE="...">	
Argument list:	*aSizeValue*	A size value as described

The JSSTag.fontSize property corresponds to the font-size CSS property. It controls the size of the text as it appears on the screen.

This is notoriously unportable and text always looks smaller on a Macintosh than it does on a Windows system. This is because the native screen resolution of a Macintosh is 72 dpi and a Windows system is 96 dpi.

You can specify the size in a variety of ways. There are four classifications for the font size value:

- Length
- Percentage
- Absolute size
- Relative size

The length value can be specified in units of measure as follows:

Points –e.g. 12pt

Millimetres –e.g. 0.4mm

Pixels –e.g. 16px

The percentage value is computed relative to the enclosing object's corresponding property (hence the cascading effect of style sheets). A percentage value is indicated with an integer followed by a percent sign, for example 120%.

The absolute size is based on the browser knowing the screen display resolution and computing some preferred font sizes. It is specified using the following keywords:

- xx-small
- x-small
- small
- medium
- large
- x-large
- xx-large

Note that some of the smaller sizes render so small on a Macintosh as to be unreadable.

The relative sizing is controlled by the two keywords:

- larger
- smaller

You can use a combination of absolute and relative sizing or any combination of sizes as this property is cascaded down through the styles.

The default value for this property is "medium".

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`Measurement` units, style.fontSize

JSSTag.fontStyle (Property)

The font attributes for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.fontStyle
	N	*myJSSTag*.fontStyle = *aStyle*
CSS syntax:	font-style: *aStyle*	
Argument list:	*aStyle*	A font presentation style

The `JSSTag.fontStyle` property corresponds to the CSS `font-style` property.

You can choose a value from these available styles:

- normal
- italic
- oblique

For some fonts, the italic and oblique fonts are simply slanted versions of the normal upright fonts. This can lead to some unattractive artefacts and so the italic form is sometimes specially designed and uses a markedly different glyph. It is intended to improve readability of italic characters.

The default value for this property is "normal".

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.fontStyle

JSSTag.fontWeight (Property)

The weight of a font for an object. This is otherwise known as the boldness.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myJSSTag*.fontWeight
	N	*myJSSTag*.fontWeight = *aWeight*
CSS syntax:	font-weight: *aWeight*	
HTML syntax:	<FONT WEIGHT="...">	
Argument list:	*aWeight*	A font weight control value

The JSSTag.fontWeight property corresponds to the font-weight CSS property.

This property controls the weight of the characters on the screen. this is sometimes called boldness although it provides a much finer level of control than simply using a bold tag. The following numerical values select a progressive weight increase:

- 100
- 200
- 300

- 400
- 500
- 600
- 700
- 800
- 900

These keywords select a pair of reasonably similar settings across platforms:

- normal
- bold

The normal setting is equivalent to 400 and the bold setting matches 700.

In addition there are two relative boldness controls:

- bolder
- lighter

Not all font families support as wide a range of possible weights as the `fontWeight` property can control. Nevertheless, the browser makes its best efforts to deliver the requested appearance.

The default value for this property is "normal".

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units, `style.fontWeight`

JSSTag.height (Property)

The height of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.height`
	N	*myJSSTag*`.height = `*aHeight*
CSS syntax:	`height: `*aHeight*	

HTML syntax:	`<... HEIGHT="...">`	
Argument list:	*aHeight*	An object height value
Event handlers:	This controls the height of an image object on screen for example.	

The `JSSTag.height` property corresponds to the `height` CSS property.

This property can be used to control the height of an object on the screen. The value can be specified as a length measured in units, a percentage of the containing block the element is located in or an "auto" value that lets the browser compute the size.

The default value for this property is "auto".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`JSSTag.width, style.height`

JSSTag.lineHeight (Property)

The line height spacing for an object. This is the distance between the baselines of two adjacent lines of text.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.lineHeight`
	N	*myJSSTag*`.lineHeight =` *aLineHeight*
CSS syntax:	`line-height:` *aLineHeight*	
Argument list:	*aLineHeight*	A spacing between lines of text

The `JSSTag.lineHeight` property corresponds to the `line-height` CSS property.

This value specifies the distance between the baselines for adjacent lines; more accurately, it indicates the minimum distance, because an object larger than the line height value will cause lines to be leaded further apart to accommodate the item within the display.

The following values are meaningful in this property:

- A numeric line height measured in points
- A value in pixels
- A distance in mm
- A percentage of the parent line height value

The default value for this depends on the user preference settings for font sizes.

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units, `style.lineHeight`

JSSTag.listStyleType (Property)

A list style bullet selector for the object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.listStyleType`
	N	*myJSSTag*`.listStyleType = `*aStyle*
CSS syntax:	`list-style-type: `*aStyle*	
HTML syntax:	`<... TYPE="...">`	
Argument list:	*aStyle*	A list style control

The `JSSTag.listStyleType` property corresponds to the `list-style-type` CSS property.

This is used to control the appearance of lists of items. It can have the following values:

- disc
- circle
- square
- decimal
- decimal-leading-zero

- lower-roman
- upper-roman
- lower-alpha
- lower-latin
- upper-alpha
- upper-latin
- lower-greek
- hebrew
- armenian
- georgian
- cjk-ideographic
- hiragana
- katakana
- hiragana-iroha
- katakane-iroha
- none

The value "none" suppresses labels on the list.

There are various graphic symbols.

You can also number the list items using different documentation conventions (numbers, roman numerals, letters etc).

There is also support for international character fonts and localization requirements.

The default value for this property is "disc".

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4 and is completely removed from Netscape 6.0.

See also:	style.listStyleType

JSSTag.marginBottom (Property)

The margin at the bottom of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.marginBottom
	N	*myJSSTag*.marginBottom = *aWidth*
CSS syntax:	margin-bottom: *aWidth*	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.marginBottom` property corresponds to the `margin-bottom` CSS property.

The content of an object is enclosed in a bounding extent rectangle. Around this is placed a padding space controlled by the padding parameters, which expands the bounding –the border is then placed around this. The border width increases the bounding rectangle around the object still further. Last of all the margin is placed around the outside. This forms a rectangle that is used to locate the object adjacent to any other objects or page margins.

The value of this property can be specified as follows:

- A length value in one of several units of measure
- A percentage of the width of the object

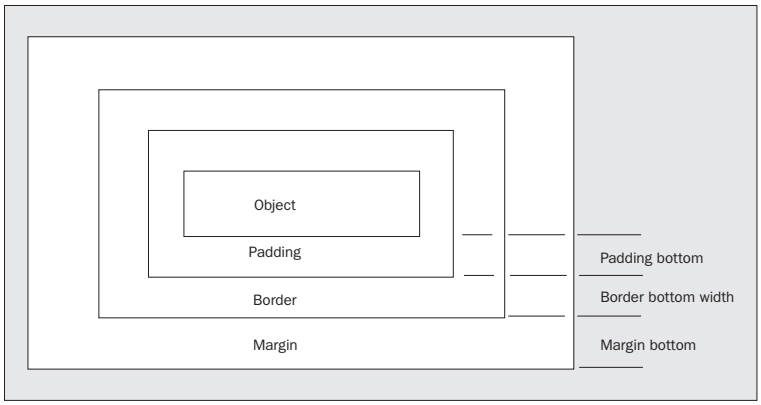
In addition a keyword can be used:

- auto

The auto setting need not be applied to all of the margins, but according to how it is used, some fairly complex margin computation is applied. You should consult one of the standard texts on CSS styles.

The initial value is set to 0 allowing for no margin value at all.

This value is not inherited from its parent container element.



Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.marginLeft (Property)

The margin at the left of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.marginLeft
	N	*myJSSTag*.marginLeft = *aWidth*
CSS syntax:	margin-left: *aWidth*	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.marginLeft` property corresponds to the `margin-left` CSS property.

Refer to the `JSSTag.marginBottom` topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.marginRight (Property)

The margin at the right of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.marginRight
	N	*myJSSTag*.marginRight = *aWidth*

CSS syntax:	`margin-right: aWidth`	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.marginRight` property corresponds to the `margin-right` CSS property.

Refer to the `JSSTag.marginBottom` topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.margins() (Method)

Set all margin values for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myJSSTag*.margins(*aTop*, *aHoriz*, *aBottom*)
	N	*myJSSTag*.margins(*aTop*, *aRight*, *aBottom*, *aLeft*)
	N	*myJSSTag*.margins(*aValue*)
	N	*myJSSTag*.margins(*aVert*, *aHoriz*)
CSS syntax:	margin: *aTop*, *aHoriz*, *aBottom*margin: *aTop*, *aRight*, *aBottom*, *aLeft*margin: *aValue*margin: *aVert*, *aHoriz*	
Argument list:	*aBottom*	A margin applied at the bottom
	aHoriz	A margin applied left and right
	aLeft	A margin applied to the left
	aRight	A margin applied to the right
	aTop	A margin applied at the top
	aValue	A margin value applied all round
	aVert	A margin applied top and bottom

This method provides a way to set all four margin values at once. The number of arguments is variable and the values are applied in different ways according to how many are supplied.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	JavaScript Style Sheets, style.margin

JSSTag.marginTop (Property)

The margin at the top of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.marginTop`
	N	*myJSSTag*`.marginTop = `*aWidth*
CSS syntax:	`margin-top: `*aWidth*	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.marginTop` property corresponds to the `margin-top` CSS property.

Refer to the `JSSTag.marginBottom` topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.paddingBottom (Property)

The padding at the bottom of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.paddingBottom`
	N	*myJSSTag*`.paddingBottom = `*aWidth*
CSS syntax:	`padding-bottom: `*aWidth*	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.paddingBottom` property corresponds to the `padding-bottom` CSS property.

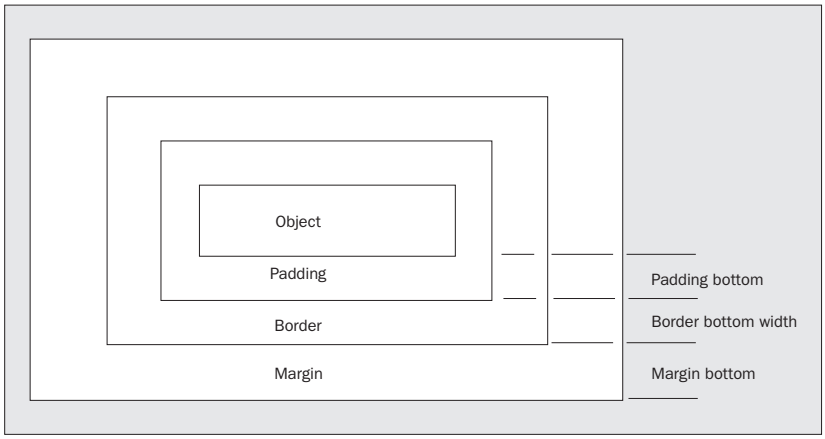
The content of an object is enclosed in a bounding extent rectangle. Around this is placed a padding space controlled by the padding parameters. The padding expands the bounding. Then the border is placed around this. The border width increases the bounding rectangle around the object still further. Last of all the margin is placed around the outside. This forms a rectangle that is used to locate the object adjacent to any other objects or page margins.

The value of this property can be specified as follows:

- A length value in one of several units of measure
- A percentage of the width of the object

The initial value is set to 0 allowing for no padding value at all.

This value is not inherited from its parent container element.



Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.paddingLeft (Property)

The padding at the left of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive Deprecated`	
JavaScript syntax:	N	*myJSSTag*.paddingLeft
	N	*myJSSTag*.paddingLeft = *aWidth*

CSS syntax:	`padding-left: aWidth`	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.paddingLeft` property corresponds to the `padding-left` CSS property.

Refer to the `JSSTag.paddingBottom` topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.paddingRight (Property)

The padding at the right of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.paddingRight
	N	*myJSSTag*.paddingRight = *aWidth*
CSS syntax:	`padding-right: aWidth`	
Argument list:	*aWidth*	A CSS length value

The *JSSTag.paddingRight* property corresponds to the *padding-right* CSS property.

Refer to the *JSSTag.paddingBottom* topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.paddings() (Method)

Set all padding values for an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myJSSTag*.paddings(*aTop*, *aHoriz*, *aBottom*)
	N	*myJSSTag*.paddings(*aTop*, *aRight*, *aBottom*, *aLeft*)
	N	*myJSSTag*.paddings(*aValue*)
	N	*myJSSTag*.paddings(*aVert*, *aHoriz*)
CSS syntax:	padding: *aTop*, *aHoriz*, *aBottom*padding: *aTop*, *aRight*, *aBottom*, *aLeftpadding*: *aValuepadding*: *aVert*, *aHoriz*	
Argument list:	*aBottom*	A padding applied at the bottom
	aHoriz	A padding applied left and right
	aLeft	A padding applied to the left
	aRight	A padding applied to the right
	aTop	A padding applied at the top
	aValue	A padding value applied all round
	aVert	A padding applied top and bottom

This method provides a way to set all four padding values at once. The number of arguments is variable and the values are applied in different ways according to how many are supplied.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.padding

JSSTag.paddingTop (Property)

The padding at the top of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myJSSTag*.paddingTop
	N	*myJSSTag*.paddingTop = *aWidth*
CSS syntax:	padding-top: *aWidth*	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.paddingTop` property corresponds to the `padding-top` CSS property.

Refer to the `JSSTag.paddingBottom` topic for details.

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units

JSSTag.rgb() (Method)

A JSS style control method.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myTags*`.`*anItem*`.rgb(`*aRed*`, `*aGreen*`, `*aBlue*`)`
Argument list:	*aBlue*	Blue intensity value
	aGreen	Green intensity value
	anItem	A tag name such as P or B or H1
	aRed	Red intensity value

The `rgb()` method provides a convenient way to define an RGB triplet in a single call.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

Example code:

```
// Some red texttags.P.rgb(255, 0, 0);
```

See also:	JavaScript Style Sheets

Cross-references:

Wrox *Instant JavaScript* –page 50

JSSTag.textAlign (Property)

The text alignment within the object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.textAlign
	N	*myJSSTag*.textAlign = *anAlignment*
CSS syntax:	text-align: *anAlignment*	
HTML syntax:	<... ALIGN="...">	
Argument list:	*anAlignment*	A text alignment control

The `JSSTag.textAlign` property corresponds to the `text-align` CSS property.

This property controls the justification of text between the left and right margins. The following values are supported:

- left
- right
- center
- justify

IN addition a string can be used to indicate a character to use for decimal alignment. This need not be a dot but can be other characters to present a neatly laid out tabular column.

The default value for this property depends on the user preference settings in the user agent.

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.textAlign

JSSTag.textDecoration (Property)

The text decoration (underline, overline etc) for text in the object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.textDecoration
	N	*myJSSTag*.textDecoration = *aDecoration*
CSS syntax:	text-decoration: *aDecoration*	
Argument list:	*aDecoration*	A list of decorations for some text

The `JSSTag.textDecoration` property corresponds to the `text-decoration` CSS property.

This controls the decoration of the text on screen with additional overprinting or dynamics.

The following values are permitted:

- underline
- overline
- line-through
- blink

Beware when you use the underline attribute, people may think that it is a hyperlink. Also avoid using the blink decoration unless you really need to. It can become very annoying for the user unless it is used with taste and discretion.

The default value for this property is "none".

This value is not inherited from its parent container element although the attributes of the parent will persist.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.textDecoration

JSSTag.textIndent (Property)

The indentation of text in the object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.textIndent
	N	*myJSSTag*.textIndent = *anIndent*
CSS syntax:	text-indent: *anIndent*	
Argument list:	*anIndent*	A value to indent the first line of text

The *JSSTag.textIndent* property corresponds to the *text-indent* CSS property.

This value is used to indent the first line of text in the styled element.

The value is numeric but can be specified in several ways:

- Absolute (10)
- Relative (-5)
- Percentage of paragraph width (10%)

The default value for this property is 0, meaning no indent at all.

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	Measurement units, style.textIndent

JSSTag.textTransform (Property)

The transformation of text in the object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated
Property/method value type:	`String primitive`

JavaScript syntax:	N	*myJSSTag*.textTransform
	N	*myJSSTag*.textTransform = *aTransformation*
CSS syntax:	text-transform: *aTransformation*	
Argument list:	*aTransformation*	A character glyph transform

The JSSTag.textTransform property corresponds to the text-transform CSS property.

This property provides a means to change the glyphs that are used without affecting the underlying source text. The following values are permitted:

- capitalize
- uppercase
- lowercase
- none

The default value is "none".

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.textTransform

JSSTag.verticalAlign (Property)

Control over the vertical alignment of the object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myJSSTag*.verticalAlign
	N	*myJSSTag*.verticalAlign = *anAlignment*
CSS syntax:	vertical-align: *anAlignment*	
HTML syntax:	<... VALIGN="...">	
Argument list:	*anAlignment*	A vertical alignment control

The `JSSTag.verticalAlign` property corresponds to the `vertical-align` CSS property.

This property controls the vertical alignment of an element. It provides a way to raise or lower individual letters to create subscript or superscript effects. The technique can be used to move images relative to a line of text.

The following values are meaningful for this property:

- baseline
- sub
- super
- top
- text-top
- middle
- bottom
- text-bottom

In addition you can specify a percentage value and a length value in various units of measure.

The default value for this property is "baseline".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	style.`verticalAlign`

JSSTag.whiteSpace (Property)

The control of white space collapsing or retention in the object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*.whiteSpace
	N	*myJSSTag*.whiteSpace = *aControl*
CSS syntax:	white-space: *aControl*	
Argument list:	*aControl*	A whitespace preservation control

The `JSSTag.whiteSpace` property corresponds to the `white-space` CSS property.

This property controls how tabs, newline characters and additional whitespace inside an element is presented. The following values are available:

- normal
- pre
- nowrap

The default value for this property is "normal".

This value is inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`style.whiteSpace`

JSSTag.width (Property)

This is the width of an object.

Availability:	CSS level –1 JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myJSSTag*`.width`
	N	*myJSSTag*`.width = `*aWidth*
CSS syntax:	`width: `*aWidth*	
HTML syntax:	`<... WIDTH="...">`	
Argument list:	*aWidth*	A CSS length value

The `JSSTag.width` property corresponds to the `width` CSS property.

This property can be used to control the width of an object on the screen. The value can be specified as a length measured in units, a percentage of the containing block the element is located in or an "auto" value that lets the browser compute the size.

The default value for this property is "auto".

This value is not inherited from its parent container element.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`JSSTag.height, style.width`

JSSTags object (Object/JSS)

Part of the Netscape Navigator style JSS rendering support.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myJSSTags* = *myDocument*.classes.*aClassName*
	N	*myJSSTags* = *myDocument*.ids.*anIdValue*
Argument list:	*aClassName*	The name of a style class
	anIdValue	The value of an ID="..." HTML tag attribute
Object properties:	<tagName>	

This object is somewhat like an array in that it contains a collection of objects that can be accessed associatively by name. However, unlike an array, it does not respond to the length property request. Also unlike an array, you cannot access its members using index values.

The only meaningful property of this object is one of its array elements corresponding to an HTML tag name. There is one item in this collection for each HTML tag.

This is part of the deprecated JSS support in Netscape 4.0. It is not recommended that you use these facilities in new projects.

You cannot enumerate this object to inspect its properties.

The `document.tags` object has properties that correspond to each of the stylable tags –for example, there is a `document.tags.P`, `document.tags.B` and `document.tags.H1` object.

Each of those objects has properties such as `borderWidth` and `color` so you can set or get the property value.

Note that the `tags` object properties can be specified in mixed case as it is case-insensitive. Its properties contain objects that correspond to HTML tags and therefore they also have case-insensitive properties that correspond to each tag's attributes.

Its an interesting way to control style from JavaScript, but since it was only ever supported in Netscape 4.0 and is no longer available in Netscape 6.0 (which fully supports CSS), there is no future for JSS.

Warnings:

- This is sometimes called `tags` object but if you inspect the object with some script that reveals its constructor, you will see it is really a member of the `JSSTags` class.
- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`Document.ids[]`, `Document.tags[]`, JavaScript Style Sheets, `JSSTag` object, `JSSTags.<tagName>`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
`<tagName>`	1.2 +	-	4.0 +	-	-	-	-	-	-	-	Warning, Deprecated

Property attributes:

DontEnum.

JSSTags.<tagName> (Property)

A JSS object corresponding to a single HTML tag.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`JSSTag object`	
JavaScript syntax:	N	*myDocument*`.classes.`*aClassName*`.`*aTagName*
Argument list:	*aClassName*	The name of a style class
	aTagName	The name of an HTML tag

This object represents an HTML tag. However it is only available in Netscape 4.0 and is a means of access that ultimately leads to a `Style` object.

You can refer to a specific HTML tag name or you can use the value "all" to indicate that all HTML tags of any kind with the appropriate CLASS name will be affected.

This mechanism is radically different to and much more complex than the simple style property belonging to an `Element` object in MSIE.

From the value in this property, you would traverse the style tree down another level to find a style object with properties that directly affect the appearance of the document.

Warnings:

- Deprecated for any further use. This was available only in Netscape 4.0 and is completely removed from Netscape 6.0.

See also:	`JSSTags` object

Jump statement (Definition)

Unconditionally jump to a new location in the script.

A jump statement is one, which forces the flow of execution to jump unconditionally to another location in the script.

Jump statements in JavaScript are used to terminate iteration statements.

A function call causes execution to go unconditionally to a new location (the beginning of the function's script source text block –its body) but a function call is not strictly a jump statement because the flow of control returns eventually to the line following the function call.

A return statement is considered to be a jump statement.

See also:	`break, continue, goto, return,` Statement

KBD object (Object/HTML)

An object representing content to be displayed as if typed on the keyboard.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myKBD* = *myDocument*.all.*anElementID*
	IE	*myKBD* = *myDocument*.all.tags("KBD")[*anIndex*]
	IE	*myKBD* = *myDocument*.all[*aName*]
	-	*myKBD* = *myDocument*.getElementById(*anElementID*)
	-	*myKBD* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myKBD* = *myDocument*.getElementsByTagName("KBD")[*anIndex*]
HTML syntax:	`<KBD> ... </KBD>`	
Argument list:	*anElementID*	The `ID` attribute of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	

The appearance of the content described by this object is likely to look similar to that enclosed in `<CODE>`, `<LISTING>` or `<PRE>` tags.

See also:	`Element` object, `LISTING` object, `PRE` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	1.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	1.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Keyboard events (Definition)

Some events within the event-handling complex of a browser are related to keyboard handling.

The following events relate to keyboard handling:

- `onKeyDown`
- `onKeyUp`
- `onKeyPress`

These events are classified as keyboard events because they are generated as a result of user interactions with the keyboard.

Note that the `onKeyPress` event is triggered as a result of a matching pair of `onKeyDown` and `onKeyUp` events. You should not rely on the events arriving in any particular order, although `onKeyPress` probably arrives before `onKeyUp`.

For example, as a key is pressed, an `onKeyDown` event is fired. As it is released, an `onKeyUp` event and an `onKeyPress` event are fired. An `onKeyPress` is only fired once for each `onKeyDown` and `onKeyUp` pair.

In Netscape, you can capture keyboard events by calling the `captureEvents()` method like this:

```
window.captureEvents(Event.KEYPRESS);
```

This need not necessarily apply to the window; the area of interest can be more limited by selecting an appropriate object.

Example code:

```
<!-- Event programming example supplied by Alex.Abacus -->
<SCRIPT LANGUAGE="JavaScript1.2">

// Object sensing routine
var isNN4up = (window.Event)? true : false;

// Key press handler designed for cross platform use
function key_press_event_handler(e)
{
   if (isNN4up)
   {
      var whichKey = e.which;
   }
   else
   {
      var whichKey = window.event.keyCode;
   }
   var realKey = String.fromCharCode(whichKey);
   window.status = 'You pressed ' + realKey +
   ' (Key code: ' + whichKey + ')'
}

// Register event handler for NNav
if (isNN4up)
{
   document.captureEvents(Event.KEYPRESS);
}

// Register event handler for MSIE
document.onkeypress = key_press_event_handler;
</SCRIPT>
```

See also:	`captureEvents()`, `Event`, Event names, Event type constants, `Event.modifiers`, `Event.which`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `String.fromCharCode()`

Keyword (Definition)

The keywords that ECMAScript defines should be avoided when you create your own identifier or variable names.

A keyword is a word that has special significance in the JavaScript language. It follows the rules that ECMA lays down for describing identifiers. All of the JavaScript keywords are reserved and define the language syntax. You must not use any of them as identifier names for variables, properties, methods and functions that you define.

ECMAScript reserves a set of keywords for future use. These are intended to make provision for future language features and to give developers warning that they should avoid using these keywords in order that their scripts should continue to operate when the language is revised.

Other special names are defined by JavaScript to identify properties of the `Global` object and constructor functions of the built-in data types. You should avoid these too, unless you are intentionally overriding their functionality with your own.

Here is a list of keywords that ECMA edition 2 mandates a compliant implementation should support:

- `break`
- `continue`
- `delete`
- `else`
- `for`
- `function`
- `if`
- `in`
- `new`
- `return`
- `this`
- `typeof`
- `var`
- `void`
- `while`
- `with`

In addition, these are constants that should also be avoided:

- `true`
- `false`
- `null`

The third edition of the ECMA standard adds these keywords which in the earlier edition were reserved for future use:

- `case`
- `catch`
- `default`
- `do`
- `finally`
- `instanceof`
- `switch`
- `throw`
- `try`

The remaining reserved keywords as of edition 3 are:

- `abstract`
- `boolean`
- `byte`
- `char`
- `class`
- `const`
- `debugger`
- `double`
- `enum`
- `export`
- `extends`
- `final`
- `float`
- `goto`
- `implements`
- `import`
- `int`
- `interface`
- `long`
- `native`
- `package`
- `private`
- `protected`
- `public`
- `short`
- `static`
- `super`
- `synchronized`
- `throws`
- `transient`
- `volatile`

However, you should note that Netscape anticipates a future standard and supports these already:

- `export`
- `import`

The JavaScript 2.0 project defines these which should also be avoided and which will likely be added to a later edition of the ECMAScript standard:

- `namespace`
- `use`

Many implementations of JavaScript will introduce additional keywords. Some will provide functional behavior for the reserved keywords. To remain ECMAScript compliant, the reserved words specified in edition 3 must be supported sufficiently to prevent parsing errors, but need not provide any meaningful functionality.

You can code defensively to avoid any future problems. Using an underscore character or digit in your identifier names should improve the chances of your script continuing to operate properly in later versions of the language. Using upper case may help, but is less of a guarantee of safety. In particular, you should be very careful to avoid the names of properties and methods belonging to the `Global` object.

See also:	Lexical element, Reserved word, Token

Cross-references:

ECMA 262 edition 2 –section –7.4.2

ECMA 262 edition 3 –section –7.5.2

Label (Definition)

An identifier marking a section of code.

Availability:	ECMAScript edition –3

In JavaScript version 1.2, the `case` and `default` labels were added, which introduced as a by-product the fact that any fragment of code can be labelled with an identifier.

The identifier can be any legal JavaScript identifier that does not match a reserved keyword.

The namespace that labels exist in, is separate to that of variables and function names. This means you can use the same identifier names over again, although it's probably good practice not to.

Adding a label in front of an iterator allows you to associate a label name with a `break` or `continue` statement. This means that you can break or continue nested iterators from deep inside them. This can greatly simplify the logic of a looping system.

If the `goto` keyword is ever implemented, it would depend on this labelling mechanism being extended to provide a useful destination.

Example code:

```
// An example break to a labelled line
outerLoop:
for (var i=0; i <= max; i++)
{
   for (var j=0; j <= max2; j++)
   {
      if (i== someNum && j ==someNum2)
      {
         break outerLoop;
      }
    }
}
```

See also:	break, Code block delimiter {}, continue, do ... while(...), for(...) ..., for(... in ...) ..., goto, Reserved Word, while(...) ...

Cross-references:

ECMA 262 edition 3 –section –12.12

Label object (Object/HTML)

Adds a legend label to an input object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	Element object	
JavaScript syntax:	IE	*myLabel* = *myDocument*.all.*anElementID*
	IE	*myLabel* = *myDocument*.all.tags("LABEL")[*anIndex*]
	IE	*myLabel* = *myDocument*.all[*aName*]
	-	*myLabel* = *myDocument*.getElementById(*anElementID*)
	-	*myLabel* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myLabel* = *myDocument*.getElementsByTagName("LABEL")[*anIndex*]
HTML syntax:	<LABEL> ... </LABEL>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	accessKey, dataFld, dataFormatAs, dataSrc, form, htmlFor, tabIndex	
Object methods:	blur(), click()	
Event handlers:	onBlur, onClick, onDblClick, onDragStart, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

MSIE supports this object type with a LABEL object, and because we often don't need to know the specific class of an object, this does not cause us any significant problems with labels. But the inconsistent object class names across browsers may need to be standardized in a more reliable way in the future.

See also:	Element object, Input object, Input.accessKey

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
dataFld	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
dataFormatAs	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
dataSrc	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
form	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
htmlFor	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
blur()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
click()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Label.htmlFor (Property)

A means of associating the label with an `Input` object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLabel*`.htmlFor`

The value of this property should correspond with the `ID="..."` HTML tag attribute for the `<INPUT>` tag that instantiates the `Input` object the label belongs to.

LANG="..." (HTML Tag Attribute)

A tag attribute that specifies the international language of some content.

This tag attribute allows the current national language to be overridden on a tag by tag basis if necessary. There are many values for international languages.

Refer to the Language codes topic for a list of language code values.

See also:	`Element.lang`, Language codes

Language codes (Definition)

Language codes are used to define locale specific handling.

The `Element.lang` property controls the locale specific text rendering. It allows special characters to be handled appropriately and special character sets to be supported properly according to the national language variants.

Here is a partial list of some example language codes to be used with it:

Code	Language	Country
af	Afrikaans	Standard
ar_AE	Arabic	U.A.E.
ar_BH	Arabic	Bahrain
ar_DZ	Arabic	Algeria
ar_EG	Arabic	Egypt
ar_IQ	Arabic	Iraq

Table continued on following page

Code	Language	Country
ar_JO	Arabic	Jordan
ar_KW	Arabic	Kuwait
ar_LB	Arabic	Lebanon
ar_LY	Arabic	Libya
ar_MA	Arabic	Morocco
ar_OM	Arabic	Oman
ar_QA	Arabic	Qatar
ar_SA	Arabic	Saudi Arabia
ar_SY	Arabic	Syria
ar_TN	Arabic	Tunisia
ar_YE	Arabic	Yemen
be	Belarusian	Standard
be_BY	Belorussian	Belarus
bg	Bulgarian	Standard
bg_BG	Bulgarian	Bulgaria
ca	Catalan	Standard
ca_ES	Catalan	Spain
cs	Czech	Standard
cs_CZ	Czech	Czech Republic
da	Danish	Standard
da_DK	Danish	Denmark
de	German	Standard
de_AT	German	Austria
de_CH	German	Switzerland
de_DE	German	Germany
de_LI	German	Liechtenstein
de_LU	German	Luxembourg
el	Greek	Standard
el_GR	Greek	Greece
en	English	Standard
en_AU	English	Australia
en_BZ	English	Belize
en_CA	English	Canada
en_GB	English	Great Britain
en_IE	English	Ireland
en_JM	English	Jamaica
en_NZ	English	New Zealand

Table continued on following page

Code	Language	Country
en_TT	English	Trinidad
en_UK	English	United Kingdom
en_US	English	United States
en_ZA	English	South Africa
es	Spanish (Traditional or modern)	Spain
es_AR	Spanish	Argentina
es_BO	Spanish	Bolivia
es_CL	Spanish	Chile
es_CO	Spanish	Colombia
es_CR	Spanish	Costa Rica
es_DO	Spanish	Dominican Republic
es_EC	Spanish	Ecuador
es_ES	Spanish	Spain
es_GT	Spanish	Guatemala
es_HN	Spanish	Honduras
es_MX	Spanish	Mexico
es_NI	Spanish	Nicaragua
es_PA	Spanish	Panama
es_PE	Spanish	Peru
es_PR	Spanish	Puerto Rico
es_PY	Spanish	Paraguay
es_SV	Spanish	El Salvador
es_UY	Spanish	Uruguay
es_VE	Spanish	Venezuela
et	Estonian	Standard
et_EE	Estonian	Estonia
eu	Basque	Standard
fa	Farsi	Standard
fi	Finnish	Standard
fi_FI	Finnish	Finland
fo	Faeroese	Standard
fr	French	Standard
fr_BE	French	Belgium
fr_CA	French	Canada
fr_CH	French	Switzerland
fr_FR	French	France
fr_LU	French	Luxembourg
gd	Gaelic	Scotland

Table continued on following page

Code	Language	Country
gd_IE	Gaelic	Ireland
he	Hebrew	Standard
hi	Hindi	Standard
hr	Croatian	Standard
hr_HR	Croatian	Croatia
hu	Hungarian	Standard
hu_HU	Hungarian	Hungary
in	Indonesian	Standard
is	Icelandic	Standard
is_IS	Icelandic	Iceland
it	Italian	Standard
it_CH	Italian	Switzerland
it_IT	Italian	Italy
iw_IL	Hebrew	Israel
ja	Japanese	Standard
ja_JP	Japanese	Japan
ji	Yiddish	Standard
ko	Korean	Johab
ko_KR	Korean	Korea
lt	Lithuanian	Standard
lt_LT	Lithuanian	Lithuania
lv	Latvian	Standard
lv_LV	Latvian	Latvia
mk	Macedonian	Standard
mk_MK	Macedonian	Macedonia
ms	Malaysian	Standard
mt	Maltese	Standard
nl	Dutch	Standard
nl_BE	Dutch	Belgium
nl_NL	Dutch	Netherlands
no	Norwegian (Bokmal or Nynorsk)	Standard
no_NO_B	Norwegian (Bokmal)	Norway
no_NO_NY	Norwegian (Nynorsk)	Norway
pl	Polish	Standard
pl_PL	Polish	Poland
pt	Portuguese	Standard
pt_BR	Portuguese	Brazil

Table continued on following page

Code	Language	Country
pt_PT	Portuguese	Portugal
rm	Rhaeto-Romanic	Standard
ro	Romanian	Standard
ro_MO	Romanian	Moldavia
ro_RO	Romanian	Romania
ru	Russian	Standard
ru_MO	Russian	Moldavia
ru_RU	Russian	Russia
sb	Sorbian	Standard
sh_SP	Serbian (Latin)	Serbia
sk	Slovak	Standard
sk_SK	Slovak	Slovakia
sl	Slovenian	Standard
sl_SI	Slovene	Slovenia
sq	Albanian	Standard
sq_AL	Albanian	Albania
sr	Serbian (Cyrillic)	Cyrillic
sr_SP	Serbian (Cyrillic)	Serbia
sv	Swedish	Standard
sv_FI	Swedish	Finland
sv_SE	Swedish	Sweden
sx	Sutu	Standard
sz	Sami	Lapland
th	Thai	Standard
tn	Tswana	Standard
tr	Turkish	Standard
tr_TR	Turkish	Turkey
ts	Tsonga	Standard
uk	Ukrainian	Standard
uk_UA	Ukrainian	Ukraine
ur	Urdu	Standard
ve	Venda	Standard
vi	Vietnamese	Standard
xh	Xhosa	Standard
zh_CN	Chinese	People's Republic of China
zh_HK	Chinese	Hong Kong, S.A.R. China
zh_SG	Chinese	Singapore
zh_TW	Chinese	Taiwan
zu	Zulu	Standard

The language codes are derived from the ISO 639 language standard and the ISO 3166 country codes standard. A language code is an ISO 639 value followed by an underscore and the ISO 3166 value.

The language code is typically presented in lower case while the country code is in upper case, although this is by no means consistent across browsers and platforms.

It is therefore probably safe to assume this value is case-insensitive.

See also:	`Anchor.hreflang`, ISO 3166, `LANG="..."`, `LINK.hreflang`, `Navigator.browserLanguage`, `Navigator.language`, `Navigator.systemLanguage`, `Navigator.userLanguage`, Portability, `Url.hreflang`

Layer object (Object/Navigator)

An object representing an HTML `<LAYER>` tag.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer* = *myDocument*.*aLayerName*
	N	*myLayer* = *myLayerArray*[*anIndex*]
HTML syntax:	`<ILAYER><LAYER>`	
Argument list:	*anIndex*	An index into the layer array
Object properties:	`above`, `background`, `below`, `bgColor`, `clip`, `document`, `hidden`, `left`, `name`, `pageX`, `pageY`, `parentLayer`, `siblingAbove`, `siblingBelow`, `src`, `top`, `visibility`, `window`, `x`, `y`, `zIndex`	
Object methods:	`load()`, `moveAbove()`, `moveBelow()`, `moveBy()`, `moveTo()`, `moveToAbsolute()`, `offset()`, `resizeBy()`, `resizeTo()`	
Functions:	`captureEvents()`, `handleEvent()`, `releaseEvents()`, `routeEvent()`	
Event handlers:	`onBlur`, `onFocus`, `onLoad`, `onMouseOut`, `onMouseOver`, `onMouseUp`	
Collections:	`layers[]`	

Each layer in Netscape is somewhat like a separate window or frame. This means it has its own document associated with it which can itself also contain layers. The individual objects cannot be positioned themselves but the layers they live in can be.

The Netscape `Layer` object has many properties that are similar to the MSIE `Style` object. However although they bear some similarities, they also have many differences. Furthermore a layer is not a style and therefore it is difficult to conveniently map one to the other and build cross-platform solutions without constructing a compatibility layer.

Note that Netscape prior to version 6.0 instantiates an absolutely positioned `<DIV>` container as a `Layer` object.

Event handling support via properties containing `function` objects was added to `Layer` objects at version 1.1 of JavaScript.

The example demonstrates how to control a scrolling panel and do it in a way that is cross browser compliant for MSIE and Netscape version 4. There are a lot of issues to deal with, not least the fact that Netscape supports layers but MSIE does not. It is necessary to use layers for scrolling in Netscape because you can only scroll windows or frames in Netscape if they have visible and active scrollbars. Also, the two browsers scroll vertically in opposing directions. If you are careful about the sizing of your objects, and relate the size of the window/frame the layer is drawn in, you can accomplish a continuous scrolling effect by duplicating items from the top of the list to the bottom. Then at an appropriate point, you can jump scroll back to the top of the list. If you do this right, it will appear as if the list is endless and scrolling gently in a continuous loop. This technique is a much simplified example taken from the Video On Demand console at http://www.bbc.co.uk/news where there is a panel showing a listing of live TV programs in a scrolling pane.

Warnings:

- If you are using layers to position items temporarily on the screen, then you should be careful when using absolute positioning. This can cause the layer to be absolutely positioned relative to the current mouse position and not the window border. The effect is that an item in the positioned layer will appear to follow the mouse as it is clicked. You may need to work out the mouse position and then calculate an offset to relocate the layer where you want it. You may be able to detect the mouse by removing focus from the layer that has the problem, or it may be necessary to create an empty layer which can be safely attached to the mouse. All of these issues are Netscape 4 specific.
- Layers are no longer supported in Netscape 6.0.

Example code:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
// Initialize globals
var theScrollValue = 0;
var theMaxScroll = 50;
// Work out what kind of browser we are on
function getBrowserType()
{
   var myUserAgent;
   var myMajor;
   myUserAgent   = navigator.userAgent.toLowerCase();
   myMajor       = parseInt(navigator.appVersion);
   if( (myUserAgent.indexOf('mozilla')    != -1) &&
       (myUserAgent.indexOf('spoofer')    == -1) &&
       (myUserAgent.indexOf('compatible') == -1) &&
       (myUserAgent.indexOf('opera')      == -1) &&
       (myUserAgent.indexOf('webtv')      == -1)
      )
   {
      if (myMajor > 3)
      {
         return "nav4";
      }
      return "nav";
```

```
    }
    if (myUserAgent.indexOf("msie") != -1)
    {
       if (myMajor > 3)
       {
          return "ie4";
       }
       return "ie";
    }
    return "other";
}
// Start the correct scroller for this browser
function startScroller()
{
    eval(getBrowserType() + "_scrollPage()");
}
// Browser specific scroller (IE)
function ie4_scrollPage()
{
    self.scrollTo(0,theScrollValue);
    theScrollValue++;

    if(theScrollValue == theMaxScroll)
    {
       theScrollValue = 0;
    }

    setTimeout("ie4_scrollPage()", 100);
}
// Browser specific scroller (Navigator)
function nav4_scrollPage()
{
    self.document.layer1.moveTo(0,-theScrollValue);
    theScrollValue++;

    if(theScrollValue == theMaxScroll)
    {
       theScrollValue = 0;
    }
setTimeout("nav4_scrollPage()", 20);
}
</SCRIPT>
</HEAD>
<BODY ONLOAD="startScroller();">
<LAYER TOP=0 LEFT=0 NAME="layer1">
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR HEIGHT=25><TD VALIGN=TOP>Headline 1<BR></TD></TR>
<TR HEIGHT=25><TD VALIGN=TOP>Headline 2<BR></TD></TR>
<TR HEIGHT=25><TD VALIGN=TOP>Headline 3<BR></TD></TR>
<TR HEIGHT=25><TD VALIGN=TOP>Headline 4<BR></TD></TR>
<TR HEIGHT=25><TD VALIGN=TOP>Headline 5<BR></TD></TR>
<TR HEIGHT=25><TD VALIGN=TOP>Headline 6<BR></TD></TR>
</TABLE>
</LAYER>
</BODY>
</HTML>
```

See also:	DIV object, Layer.siblingAbove, Layer.siblingBelow, LayerArray object, style object (2)

Property	JavaScript	JScript	N	IE	Opera	Notes
above	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated
background	1.2 +	-	4.0 +	-	-	Warning, Deprecated
below	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated
bgColor	1.2 +	-	4.0 +	-	-	Warning, Deprecated
clip	1.2 +	-	4.0 +	-	-	Warning, Deprecated
document	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated
hidden	1.2 +	-	4.0 +	-	-	Warning, Deprecated
left	1.2 +	-	4.0 +	-	-	Warning, Deprecated
name	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated
pageX	1.2 +	-	4.0 +	-	-	Warning, Deprecated
pageY	1.2 +	-	4.0 +	-	-	Warning, Deprecated
parentLayer	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated
siblingAbove	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated
siblingBelow	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated
src	1.2 +	-	4.0 +	-	-	Warning, Deprecated
top	1.2 +	-	4.0 +	-	-	Warning, Deprecated
visibility	1.2 +	-	4.0 +	-	-	Warning, Deprecated
window	1.2 +	-	4.0 +	-	-	Warning, Deprecated
x	1.2 +	-	4.0 +	-	-	Warning, Deprecated
y	1.2 +	-	4.0 +	-	-	Warning, Deprecated
zIndex	1.2 +	-	4.0 +	-	-	Warning, Deprecated

Method	JavaScript	JScript	N	IE	Opera	Notes
load()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
moveAbove()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
moveBelow()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
moveBy()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
moveTo()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
moveToAbsolute()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
offset()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
resizeBy()	1.2 +	-	4.0 +	-	-	Warning, Deprecated
resizeTo()	1.2 +	-	4.0 +	-	-	Warning, Deprecated

Event name	JavaScript	JScript	N	IE	Opera	Notes
onBlur	1.2 +	-	4.0 +	-	-	Warning
onFocus	1.2 +	-	4.0 +	-	-	Warning
onLoad	1.2 +	-	4.0 +	-	-	Warning
onMouseOut	1.2 +	-	4.0 +	-	-	Warning
onMouseOver	1.2 +	-	4.0 +	-	-	Warning
onMouseUp	1.2 +	-	4.0 +	-	-	Warning

Layer() (Constructor)

A means of creating new layers in Netscape.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	`new Layer()`

This constructor function should be used with the `new` operator to instantiate a new layer when needed. This provides a way to create new layers without needing to describe them in the HTML document source.

Warnings:

- No longer supported in Netscape 6.0.

Layer.above (Property)

The layer immediately above the receiving `layer` object, (or `null`, if this is the highest layer).

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Layer object`	
JavaScript syntax:	N	`myLayer.above`

Warnings:

- No longer supported in Netscape 6.0.

Property attributes:

`ReadOnly`.

Layer.background (Property)

The `background` object of a layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Background object`	
JavaScript syntax:	N	*myLayer*`.background`

If a background image is available, then its URL is contained in the `src` property of the object. Changing the `Layer.background.src` property value will replace the background with a new one. However, there may be a perceptible delay while the new image is fetched from the web server.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Background` object

Layer.below (Property)

The layer immediately below the receiving `layer` object.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Layer object`	
JavaScript syntax:	N	*myLayer*`.below`

The layers area arranged into a hierarchy and presented to the user as a stack viewed from one end. Given that you are referencing any particular layer, if there is one below it in the stack, this property will yield an object that encapsulates it.

Warnings:

- There are various intermediate layers created by the browser that may be placed into the layer hierarchy without you having put `<LAYER>` tags into the document.
- No longer supported in Netscape 6.0.

Property attributes:

`ReadOnly`.

Layer.bgColor (Property)

The background color of a layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myLayer*`.bgColor`

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.The color can be set using a Hexadecimal triplet, or a plain language color value. The default is a transparent background.

Warnings:

- No longer supported in Netscape 6.0.

See also:	Color names, Color value

Layer.captureEvents() (Function)

Part of the Netscape 4 event propagation complex.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myLayer*`.captureEvents(`*anEventMask*`)`
Argument list:	*anEventMask*	A mask constructed with the manifest event constants

This is part of the event management suite which allow events to be routed to handlers other than just the one that defaults to being associated with an event.

The events to be captured are signified by setting bits in a mask.

This method allows you to specify what events are to be routed to the receiving `Layer` object.

The events are specified by using the bitwise OR operator (|) to combine the required event mask constants into a mask that defines the events you want to capture. Refer to the Event Type Constants topic for a list of the event mask values.

A limitation of this technique is that ultimately, only 32 different kinds of events can be combined in this way and this may limit the number of events the browser can support. Since this is only supported by Netscape, the functionality is likely to be deprecated when the standards bodies agree on a standard way of handling events. Then we simply need to wait for the browser manufacturers to support the standardized behavior.

In the meantime, we shall have to implement scripts using this capability if we need to build complex event handling systems. A different script will be required for MSIE.

You may be able to factor your event handler so that you only have to make platform specific event dispatchers and can call common handling routines that can be shared between MSIE and Netscape.

Warnings:

- Since a bit mask is being used, this must be an `int32` value. This suggests that there can only be 32 different Event types supported by this event propagation model.
- This capability is deprecated and is not supported in Netscape 6.0. It never was supported by MSIE which implements a completely different event model. As it turns out, the DOM level 2 event model converges on the MSIE technique.

See also:	`captureEvents()`, `Document.captureEvents()`, `Document.releaseEvents()`, `Element.onevent`, Event propagation, Event type constants, `Frame` object, `Layer.releaseEvents()`, `onMouseMove`, `Window` object, `Window.captureEvents()`, `Window.releaseEvents()`

Layer.clip (Property)

An object that represents the `clip` region within a layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Rect object`	
JavaScript syntax:	N	*myLayer*`.clip`

This object represents a clipping rectangle that the visible part of a `display` object is viewed through. This is most likely used with a `layer` object. The layer contents would be drawn off-screen and then that part which falls within the clipping rectangle would be displayed in the window.

This can be useful for performing wipes and making parts of a layer progressively visible within some kind of transition loop.

This object exposes the following properties for accessing `Layer` clip rectangles:

- `clip.bottom`
- `clip.height`
- `clip.left`

- `clip.right`
- `clip.top`
- `clip.width`

If you are developing scripts that only run in MSIE, there is a rather nice collection of filters that can provide these transitions very elegantly. It's a pity they aren't available in other browsers. But then layers and clip rectangles aren't supported in the same way in MSIE so at least there seems to be some alternative approach to creating visual effects on each platform.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Clip` object, `Rect` object

Layer.clip.bottom (Property)

The bottom edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.clip.bottom`
	N	*myRect*`.bottom`

This defines the bottom edge of the clip region. You could modify this in a loop to create a vertical downwards wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Clip.bottom`, `Rect.bottom`

Layer.clip.height (Property)

The height of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.clip.height`
	N	*myRect*`.height`

The clip region is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Clip.height,Rect.height`

Layer.clip.left (Property)

The left edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.clip.left`
	N	*myRect*`.left`

This defines the left edge of the clip region. You could modify this in a loop to create a horizontal wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Clip.left,Rect.left`

Layer.clip.right (Property)

The right edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.clip.right`
	N	*myRect*`.right`

This defines the right edge of the clip region. You could modify this in a loop to create a horizontal wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Clip.right, Rect.right`

Layer.clip.top (Property)

The top edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.clip.top`
	N	*myRect*`.top`

This defines the top edge of the clip region. You could modify this in a loop to create a vertical upward wipe transition effect.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Clip.top, Rect.top`

Layer.clip.width (Property)

The width of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.clip.width`
	N	*myRect*`.width`

The clip region is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Clip.width`, `Rect.width`

Layer.document (Property)

The `document` object containing this layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Document object`	
JavaScript syntax:	N	*myLayer*`.document`

This is the document contained within a Netscape `layer` object. Because each layer has its own separate document, you can use the `document.open()`, `document.close()` and `document.write()` methods to operate on it.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Document` object, `Element.document`

Property attributes:

`ReadOnly`.

Layer.handleEvent() (Function)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myLayer`*`.handleEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`handleEvent(),Layer.routeEvent()`

Layer.hidden (Property)

A deprecated property that indicates whether a `layer` is hidden or not.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*`myLayer`*`.hidden`

You should not use this property in new projects. If you encounter it in existing ones, if you have the time you should remove it and use the `visibility` property instead.

Warnings:

- This property is deprecated in favor of the `Layer.visibility` property. However, even though it is deprecated some versions of Netscape require that it is forcibly set to `false` otherwise the `visibility` property will not function correctly.
- If an object is hidden, the gap will not be filled by any surrounding content.
- No longer supported in Netscape 6.0.

See also:	`Layer.visibility`

Layer.layers[] (Collection)

A deprecated property providing a list of child layers within this layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`LayerArray object`	
JavaScript syntax:	N	*myLayer*`.layers`

You should access this collection via the `document` object associated with this layer. That is the recommended technique for new projects. If you see this in an old project, then it should be removed if possible and replaced with the preferred means of access.

Warnings:

- This property is deprecated in favor of using `Layer.document.layers` in its place.
- No longer supported in Netscape 6.0.

See also:	`Document.layers[]`, `LayerArray` object

Property attributes:

`ReadOnly`.

Layer.left (Property)

The x-coordinate relative to the containing `layer`.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.left`

In Netscape this defines the left coordinate of a layer. It corresponds to the `pixelLeft` property of an MSIE `style` object.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`style.pixelLeft`, `style.posLeft`

Layer.load() (Method)

A method to load a new URL and resize a `layer`.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*`.load(`*aURL, aWidth*`)`
Argument list:	*aURL*	A document to load into the layer
	aWidth	A new width value for the layer

This is an alternative way to load a new document into a layer. You can accomplish something similar by assigning a new value to the `src` property. This method will adjust the width of the layer at the same time.

This can work very effectively if you generate dynamic content via a `javascript:` URL as the value passed to the `load()` method.

In fact this works so much better, that the `Layer.src` property can be ignored for most purposes. This seems to work well with an absolutely positioned `<DIV>` container.

Warnings:

- This doesn't work for `ILayer` elements in Netscape 4.
- No longer supported in Netscape 6.0.

See also:	`javascript:` URL, `Layer.src`

Layer.moveAbove() (Method)

Adjusts the Z ordering of a layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*`.moveAbove(`*aLayer*`)`
Argument list:	*aLayer*	The `layer` object to be moved above

This method provides a way to move a layer above another layer without needing to do complex Z index computations.

Warnings:

- No longer supported in Netscape 6.0.

Layer.moveBelow() (Method)

Adjusts the Z ordering of a `layer`.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*.moveBelow(*aLayer*)
Argument list:	*aLayer*	The `layer` object to be moved below

This method provides a way to move a layer behind another layer without needing to do complex Z index computations.

Warnings:

- No longer supported in Netscape 6.0.

Layer.moveBy() (Method)

Adjusts the X,Y position of a `layer`.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*.moveBy(*anX, aY*)
Argument list:	*anX*	A relative distance in the X axis
	aY	A relative distance in the Y axis

This method provides a way to locate a Netscape layer to a new position relative to its old one, by specified pixel amounts along both axes. Positive values move to the right or up, and negative values to the left or down.

Warnings:

- No longer supported in Netscape 6.0.

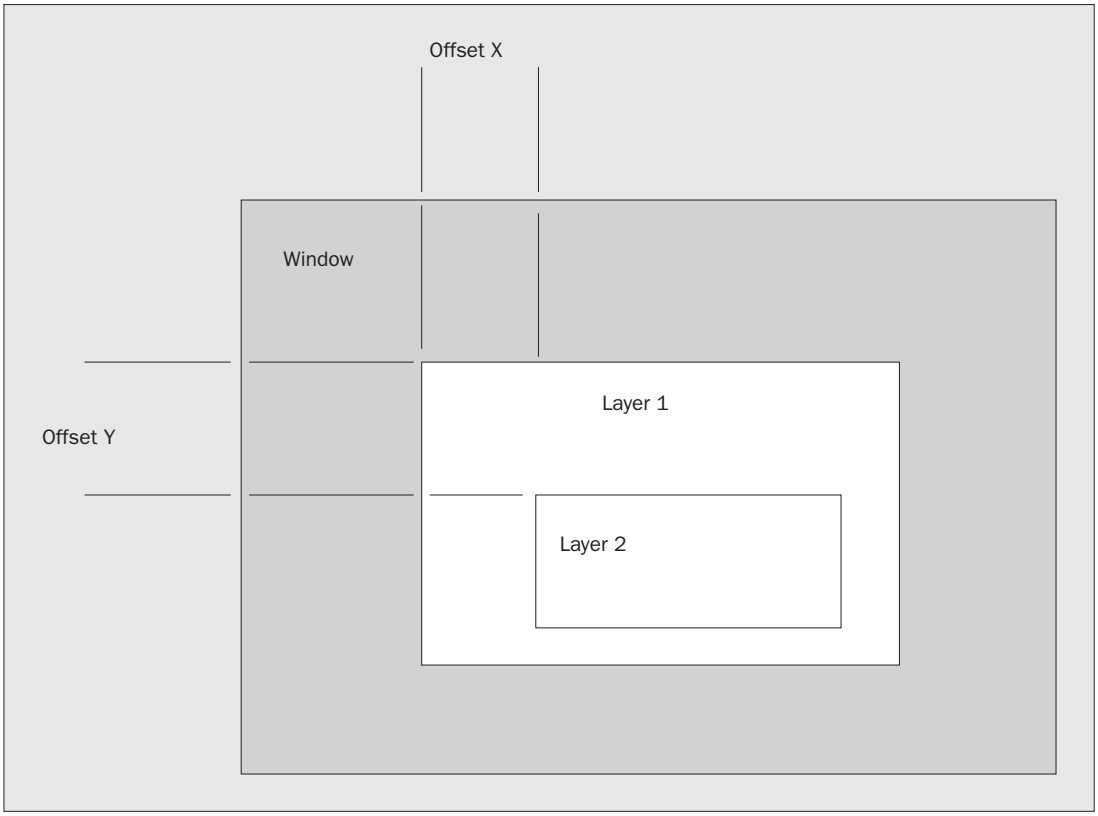
See also:	`Layer.offset()`

Layer.moveTo() (Method)

Adjust the X,Y position of a layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*.moveTo(*anX, aY*)
Argument list:	*anX*	An X coordinate relative to the parent element
	aY	A Y coordinate relative to the parent element

This method provides a way to locate a Netscape layer to a new absolute position relative to its containing parent object.



Warnings:

- No longer supported in Netscape 6.0.

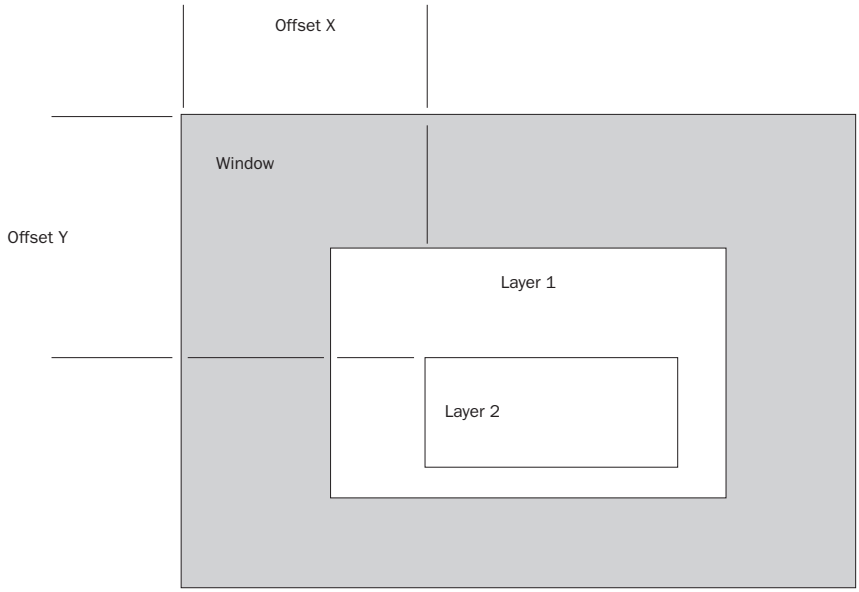
See also:	Layer.parentLayer

Layer.moveToAbsolute() (Method)

Adjust the X,Y position of a layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	`myLayer.moveToAbsolute(anX, aY)`
Argument list:	`anX`	An X coordinate relative to the page
	`aY`	A Y coordinate relative to the page

This method provides a way to locate a Netscape layer to a new absolute position relative to the document window.



Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.pageX`, `Layer.pageY`

Layer.name (Property)

This corresponds to the NAME attribute of the <LAYER> tag.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myLayer*.name
HTML syntax:	<LAYER NAME="aName">	
Argument list:	*aName*	A name for the layer object

Objects are identified either by the NAME="..." HTML tag attribute or by the ID="..." HTML tag attribute.

Netscape shows a marginal preference for the name property while MSIE seems slightly better disposed towards the ID property.

W3C comes down fairly and squarely on the ID property as being the favorite.

However in many cases, both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

Warnings:

- No longer supported in Netscape 6.0.

Property attributes:

ReadOnly.

Layer.offset() (Method)

A deprecated means of moving layers.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*.offset(*anX, aY*)
Argument list:	*anX*	A relative distance in the X axis
	aY	A relative distance in the Y axis

You should now use the moveBy() method to change the layer ordering. This method should be replaced when possible.

Warnings:

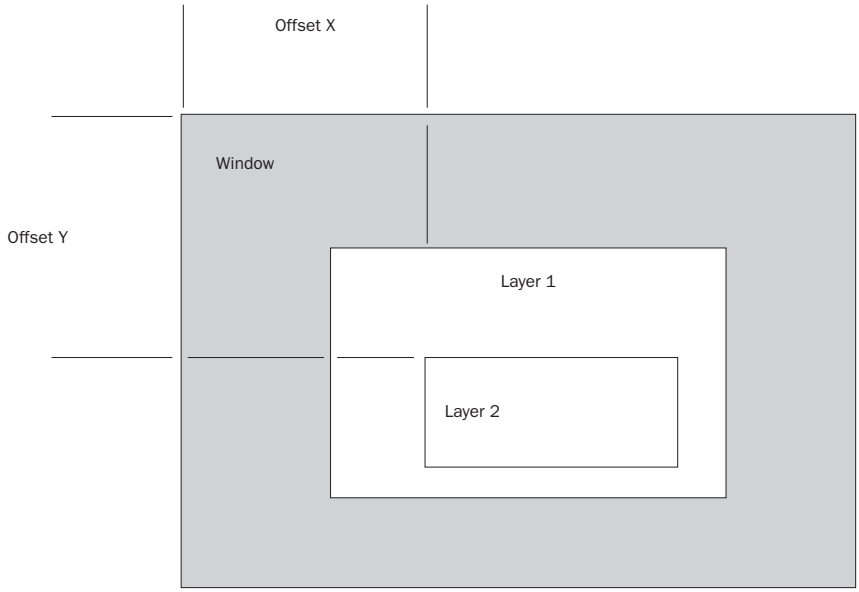
- This method is deprecated in favor of the `Layer.moveBy()` method in Netscape 4.
- No longer supported in Netscape 6.0.

See also:	`Layer.moveBy()`

Layer.pageX (Property)

The X-coordinate of the layer relative to the top level document.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.pageX`



Warnings:

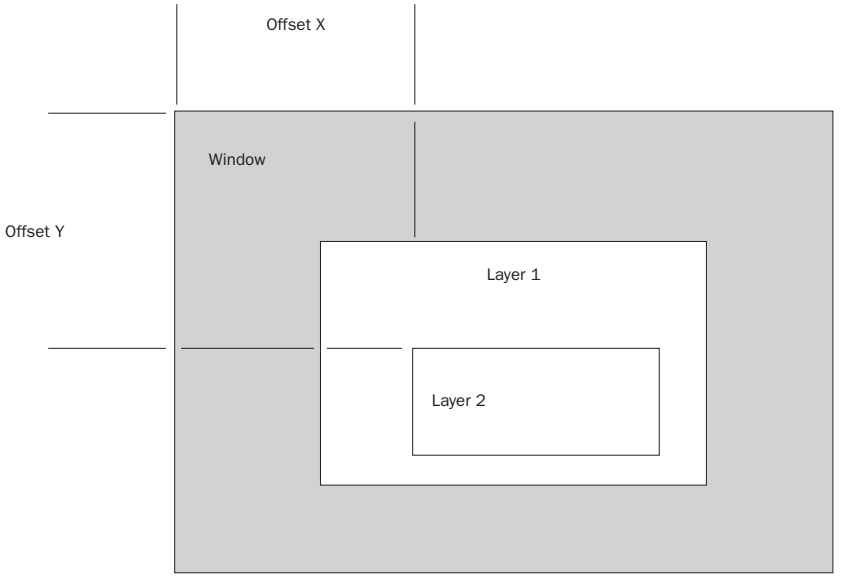
- No longer supported in Netscape 6.0.

See also:	`Layer.moveToAbsolute()`

Layer.pageY (Property)

The Y-coordinate of the layer relative to the top level document.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.pageY`



Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.moveToAbsolute()`

Layer.parentLayer (Property)

The layer containing the current layer (for a single layer, this is the `window` object).

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Layer or Window object`	
JavaScript syntax:	N	*myLayer*`.parentLayer`

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.moveTo()`

Property attributes:

`ReadOnly`.

Layer.releaseEvents() (Function)

Part of the Netscape 4 event propagation complex.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myLayer`*`.releaseEvents(`*`anEventMask`*`)`
Argument list:	*`anEventMask`*	A mask defined with the manifest event constants

This is part of the event management suite which allows events to be routed to handlers other than just the one that defaults to being associated with an event.

The events to be captured are signified by setting bits in a mask.

This method provides a means of indicating which events are no longer needing to be captured by the receiving `Layer` object.

The events are specified by using the bitwise OR operator (`|`) to combine the required event mask constants into a mask that defines the events you want to capture. Refer to the Event Type Constants topic for a list of the event mask values.

Since this is only supported by Netscape, the functionality is likely to be deprecated when the standards bodies agree on a standard way of handling events. In the meantime, we shall have to implement scripts using this capability if we need to build complex event handling systems. A different script will be required for MSIE.

You may be able to factor your event handler so that you only have to make platform specific event dispatchers, and can call common handling routines that can be shared between MSIE and Netscape.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`captureEvents()`, `Document.captureEvents()`, `Document.releaseEvents()`, `Element.onevent`, Event propagation, Event type constants, `Event.modifiers`, `Frame` object, `Layer.captureEvents()`, `onMouseMove`, `Window` object, `Window.releaseEvents()`

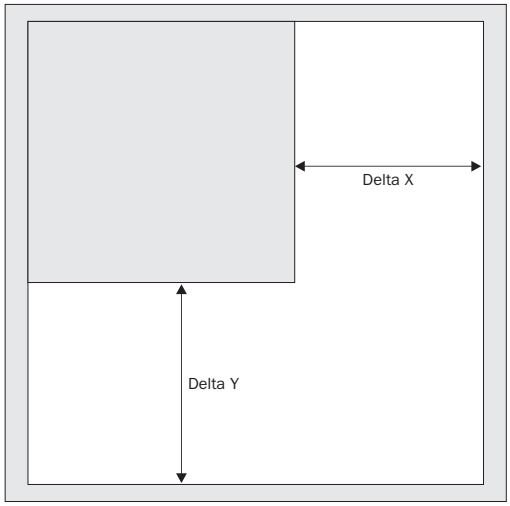
Layer.resizeBy() (Method)

Adjusts the size of a layer by a relative amount.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*.resizeBy(*anX, aY*)
Argument list:	*anX*	A relative distance in the X axis
	aY	A relative distance in the Y axis

The layer can have its size adjusted by a value measured in `pixels`. This can apply to the horizontal or vertical axis or both if necessary. The value can be positive or negative causing the layer to grow or shrink respectively.

Although the size changes, the top left corner will remain pinned to its current location growing or shrinking the layer to the right and bottom of its extent.



Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.resizeTo()`

Layer.resizeTo() (Method)

Adjusts the size of a layer to an absolute value.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayer*.resizeTo(*anX, aY*)
Argument list:	*anX*	An X axis value
	aY	A Y axis value

This method provides a way to set the absolute size of a layer as opposed to the relative sizing offered by the `resizeBy()` method.

Although the size changes, the top left corner will remain pinned to its current location growing or shrinking the layer to the right and bottom of its extent.

Warnings:

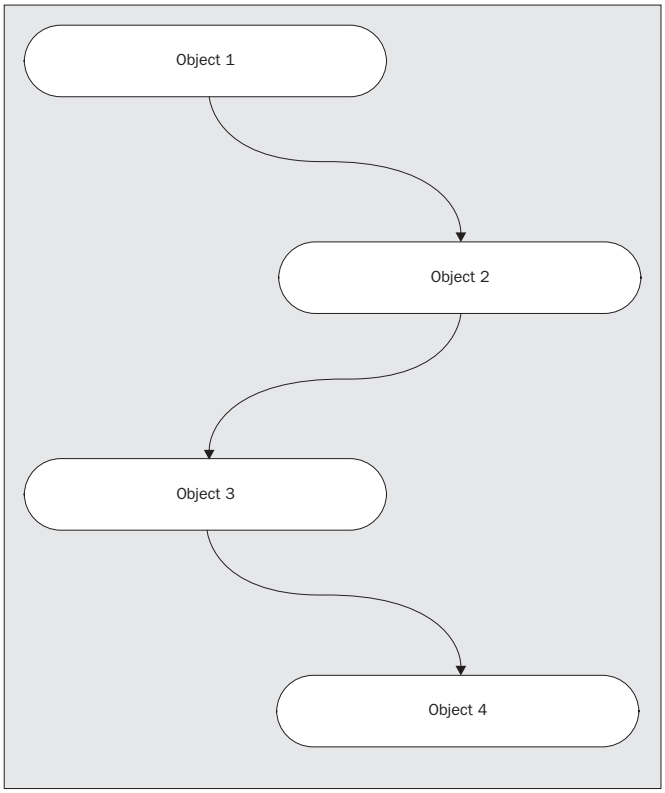
- No longer supported in Netscape 6.0.

See also:	`Layer.resizeBy()`

Layer.routeEvent() (Function)

Part of the Netscape event propagation complex.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myLayer*.routeEvent(*anEvent*)
Argument list:	*anEvent*	An `event` object



Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer.handleEvent(),Window.routeEvent()`

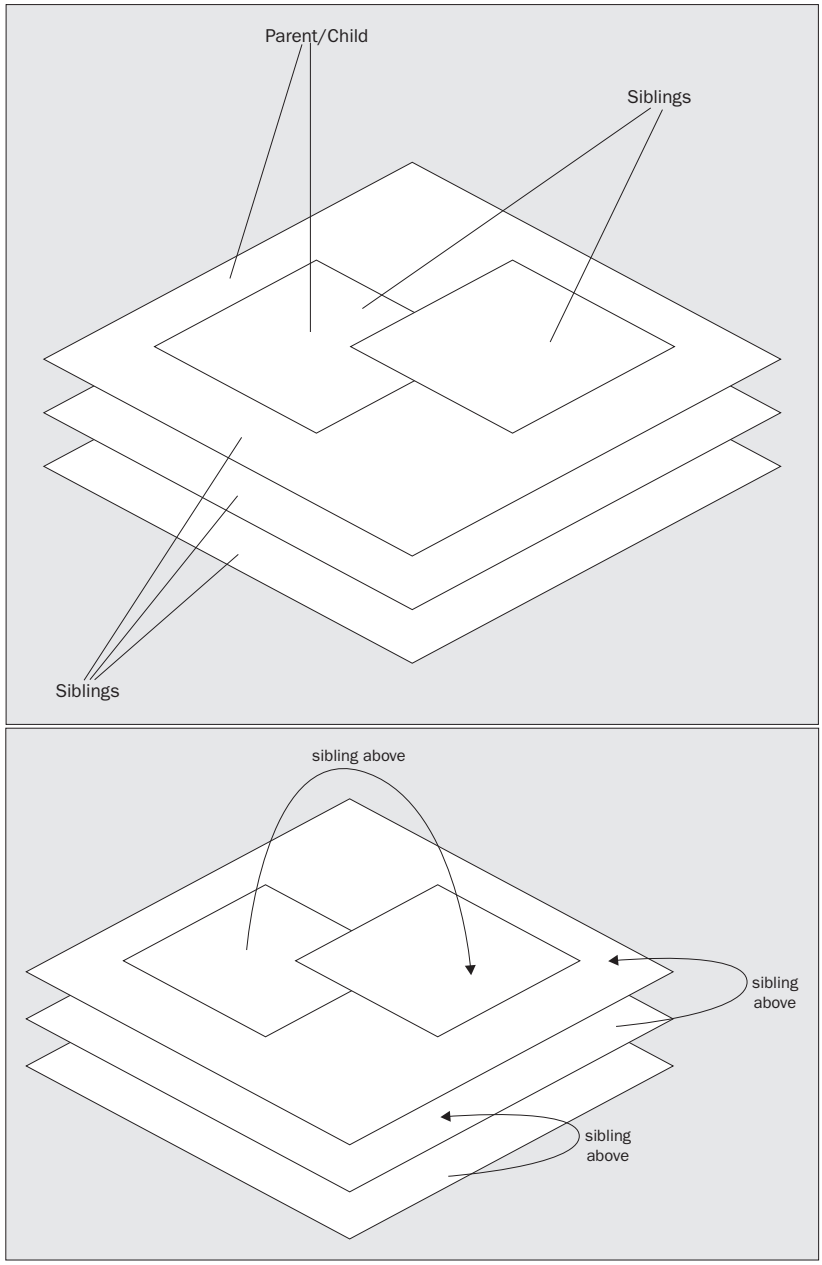
Layer.siblingAbove (Property)

A layer that is related to this layer and above it in the Z ordering.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Layer object`	
JavaScript syntax:	N	*myLayer*`.siblingAbove`

Given that a layer arrangement is yet another example of a tree structured hierarchy within a page, this property yields the next higher sibling belonging to the same parent (or `null` if this is the highest element).

Any layers sharing a common parent layer will all be present in the same `layers[]` collection. They are considered to be siblings and can be ordered with respect to one another. They are ordered collectively by their parent's relationship with its siblings and you cannot interleave siblings from different sets with one another; the set as a whole must be ordered within its parent's z-axis location.



Warnings:

- No longer supported in Netscape 6.0.

See also:	Hierarchy of objects, `Layer` object, `Layer.siblingBelow`

Property attributes:

`ReadOnly`.

Layer.siblingBelow (Property)

A layer that is related to this layer and below it in the Z ordering.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Layer object`	
JavaScript syntax:	N	*myLayer*`.siblingBelow`

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Layer` object, `Layer.siblingAbove`

Property attributes:

`ReadOnly`.

Layer.src (Property)

The source URL for a layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myLayer*`.src`

Changing this layer property value forces the browser to load a new document into the layer.

This can work very effectively if you generate dynamic content via a `javascript:` URL as the value assigned to the `src` property.

Warnings:

- If you are having problems with this, try creating an absolute positioned `<DIV>` container and then use the `Layer.load()` method instead. According to reports from other developers, that works better in Netscape 4.
- This doesn't work with `ILAYER` elements, in Netscape 4.
- No longer supported in Netscape 6.0.

See also:	`javascript: URL`, `Layer.load()`

Layer.top (Property)

The y-coordinate (in pixels) of the layer relative to its containing layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*`.top`

In Netscape this defines the top edge coordinate of a layer. It corresponds to the `pixelTop` property of an MSIE `style` object.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`style.pixelTop`, `style.posTop`

Layer.visibility (Property)

Whether a layer is visible or not.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myLayer*`.visibility`
	N	*myLayer*`.visibility = "hide"`
	N	*myLayer*`.visibility = "show"`

This controls the visibility of a Netscape layer.

It accepts the following values:

- `hide`
- `inherit`
- `show`

It also accepts these CSS syntax values (which it converts internally):

- `hidden`
- `visible`

Warnings:

- When set to `hide`, the gap won't be filled by surrounding content.
- No longer supported in Netscape 6.0.

See also:	`Layer.hidden`, `style.visibility`

Layer.window (Property)

The window that this layer belongs to.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Window object`	
JavaScript syntax:	N	*myLayer*`.window`

Warnings:

- No longer supported in Netscape 6.0.

See also:	`Frame` object, `Window` object

Layer.x (Property)

The present X position of this layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*.x

The horizontal position of the object in the display measured in `pixels`. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Warnings:

- No longer supported in Netscape 6.0.

Layer.y (Property)

The present Y position of this layer.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*.y

The vertical position of the object in the display measured in `pixels`. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Warnings:

- No longer supported in Netscape 6.0.

Layer.zIndex (Property)

The location of the layer within the Z ordered list of layers.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLayer*.zIndex

In Netscape, this is a means of ordering the layers front to back and works identically to the `zIndex` property of the MSIE `Style` object.

Warnings:

- No longer supported in Netscape 6.0.

See also:	`style.zIndex`

LayerArray object (Object/Navigator)

An array containing a list of layers in the document.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
JavaScript syntax:	N	*myLayerArray* = *myDocument*.layers
Object properties:	`length`	

Each item in this array corresponds to a `<LAYER>` tag in the document. This array also includes layers that are created in Netscape by setting the `position` attribute of an HTML `<DIV>` tag to `absolute`.

The layers in this array are ordered according to the order in which they appear in the document. Layers can be accessed associatively if they have been given an `ID` with the `ID="..."` or `NAME="..."` tag attribute. This means you can refer to an element whose `ID` is set to `ABC` by its unique name either as `document.ABC` or `document.layers["ABC"]`.

Warnings:

- There is a bug in the layer management code in Netscape. If a `<LAYER>` tag is placed into the document without an `ID="..."` or `NAME="..."` HTML tag attribute, it will increment the length count for the `LayerArray` but an object will not be placed into the array.
- Now if you try to enumerate through all the layers in the array using the `length` value, your enumeration loop will cause errors when it tries to access elements beyond the physical length of the array.
- To avoid this, you should always add `NAME="..."` HTML tag attributes to the `<LAYER>` tags to ensure the layers are stored in the array. `ID="..."` HTML tags are important and helpful when trying to access objects in MSIE and in Netscape 6.0.
- No longer supported in Netscape 6.0.

See also:	`Collection` object, `DIV` object, `Document.layers[]`, `Layer` object, `Layer.layers[]`

Property	JavaScript	JScript	N	IE	Opera	Notes
length	1.2 +	-	4.0 +	-	-	Warning, ReadOnly, Deprecated

LayerArray.length (Property)

This value reflects the number of <LAYER> tags that are in the document.

Availability:	JavaScript –1.2 Netscape –4.0 Deprecated	
Property/method value type:	Number primitive	
JavaScript syntax:	N	*myDocument*.layers.length

Warnings:

- No longer supported in Netscape 6.0.

See also:	Collection.length

Property attributes:

ReadOnly.

.lck (File extension)

Netscape configuration file (old style).

See also:	netscape.lck, Preferences

Left shift (Operator/bitwise)

A leftwards shift of a bit pattern.

Refer to:

Bitwise shift left (<<)

Left-Hand-Side expression (Definition)

Left values are the destination of an assignment.

Availability:	ECMAScript edition –2

This kind of expression identified the destination of an assignment (even if that assignment operation is only implied). Sometimes these are called LValues.

Left-hand-side expressions comprise the following:

- Variables
- Property accessors
- `new` operator
- Argument lists

See also:	Argument list, Expression, Function, `function( ... ) ...`, LValue, `new`, Property accessor

Cross-references:

ECMA 262 edition 2 –section –11.2

ECMA 262 edition 3 –section –11.2

Legend object (Object/HTML)

The `Legend` object relates to a field-set within a form.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0 Deprecated	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myLegend* = *myDocument*.all.*anElementID*
	IE	*myLegend* = *myDocument*.all.tags("LEGEND")[*anIndex*]
	IE	*myLegend* = *myDocument*.all[*aName*]
	-	*myLegend* = *myDocument*.getElementById(*anElementID*)
	-	*myLegend* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myLegend* = *myDocument*.getElementsByTagName("LEGEND")[*anIndex*]

HTML syntax:	<LEGEND> ... </LEGEND>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	accessKey, align, form, padding, tabIndex	
Event handlers:	onBlur, onChange, onClick, onDblClick, onDragStart, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onScroll, onSelectStart	

This is the legend that is associated with the field-set. It must be placed immediately inside the <FIELDSET> containing HTML tags in the document source.

Now that as CSS has become more widely available and is capable of doing the same thing, the Legend object has become deprecated.

See also:	Element object, FIELDSET object, Form object, Input object, Input.accessKey

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Deprecated
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, Deprecated
form	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Deprecated
padding	-	3.0 +	-	4.0 +	-	-	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Deprecated

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onChange	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-
onClick	1.5 +	3.0 +	6.0 +	3.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Legend.align (Property)

The alignment of the `legend` object relative to its surrounding objects.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`mylegend`*`.align`

The alignment of the `Legend` object with respect to its containing parent object is defined in this property. The expected and widely available set of alignment specifiers are:

- `absbottom`
- `absmiddle`
- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

Warnings:

- Note that there are sometimes problems with aligning `Legend` objects using the top and bottom values in MSIE version 4.

Legend.padding (Property)

The padding around a `Legend` object.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`mylegend`*`.padding`

The padding around the legend separates it from the container in much the same way as cells in a table are padded apart from one another.

The value can be specified as a floating point value followed by a unit of measure indicator. These indicators are valid:

- `cm`
- `mm`
- `in`
- `pt`
- `pc`
- `px`

The `em` and `ex` relative unit of measure are also available as well as a percentage value measured against the width of the parent container.

See also:	Measurement units

length (Property)

The length, magnitude, or size of an object depending on its type.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`frames.length`
	-	`length`
	-	*`myObject`*`.length`
	-	*`myWindow`*`.frames.length`
	-	*`myWindow`*`.length`

Many objects respond to the `length` property accessor message. They don't all yield the same sort of value, although the value they yield will always be numeric.

For example, `Window` objects have a `length` property. In this case it will yield the number of frames in the window, and it is also available as a global variable called `length`.

The other objects that provide a `length` property don't reflect its value into so many alternative ways of accessing it. Apart perhaps from the `Form.length` value which is the same as the `Form.elements.length` value. This is necessary because the `Form.elements` collection is superimposed on the `Form` object so that you can access individual elements of the form as if they were direct member properties of the `Form` object. If you try to enumerate the `Form` object, you get lots of properties enumerated which are not form elements. You should therefore enumerate the `Form.elements` array.

This table lists objects which have a `length` property, and describes what it counts.

Object	Length value
`AnchorArray`	The number of named anchors in a document
`AppletArray`	The number of applets installed in a document
`Arguments`	The number of arguments presented to a function when it was called
`Array`	One greater than the highest numbered index in an array
`Attributes`	The size of the complete set of HTML tag attributes defined for an HTML `Element` object
`CharacterData`	The length in characters of a DOM character data node
`Collection`	One greater than the highest numbered index in a collection
`Date`	The number of arguments expected (always 7)
`EmbedArray`	The number of plugins installed into a document
`Filters`	The number of filters stacked up on a displayable object in MSIE

Table continued on following page

Object	Length value
Form	The number of input elements in a form
Form.elements	The number of input elements in a form
FormArray	The number of forms in a document
FormElementsArray	The number of input elements in a form
FrameArray	The number of frames in a frameset
Frames	The number of frames in a frameset, window or iframes in a document
Function	The number of arguments expected by a function
History	The length of the historical record of pages visited
ImageArray	The number of images in a document
JavaArray	The length of a Java array encapsulated by JavaScript
JavaObject	The number of MIME types supported by a plugin encapsulated in a JavaObject
LayerArray	The number of layers in a document
LinkArray	The number of HREF links in a document
MimeTypeArray	The number of discrete MIME types that the browser supports
NamedNodeMap	The number of named DOM nodes in a node map.
NodeList	The size of a DOM node collection
OptionsArray	The range of options in a popup menu
Plugin	The number of MIME types supported by a plugin
PluginArray	The variety of plugins currently installed in the browser
ScriptArray	The number of script blocks in a document
Select	The range of options in a popup menu
SelectorArray	The number of rules in a style sheet
String	The length of a string in characters
style	The number of individual style components in a style rule
styleSheets	The number of style sheets called in by the document
textNode	The length of a textNode in characters
Window	The number of frames in a window

Although the `length` property is marked here as `ReadOnly`, in some circumstances it may be useful to assign a value to it to extend, or truncate an `Array` object.

See also:	`Array.length`, `Form.elements.length`, `Form.length`, `Window.length`

Property attributes:

`ReadOnly`.

Length units (Definition)

A sub-set of the available set of measurement units used with CSS style sheets and `style` object properties.

See also:	`Legend.padding`, `style.layoutGridChar`, `style.layoutGridLine`, Measurement units

Less than (<) (Operator/relational)

Compare two operands to determine which is nearer to -Infinity.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*anOperand1* `<` *anOperand2*
Argument list:	*anOperand1*	A value that can be compared numerically or lexically
	anOperand2	A compatible value

Returns `true` if the left operand is numerically less than the right operand or is sorted earlier in the Unicode collating sequence when two string values are compared.

In numeric comparisons, the presence of `NaN` in either or both operands will yield undefined instead of `true` or `false`.

When comparing two strings, a prefixing plus sign is present, then a numeric coercion of a string takes place before the comparison. Numeric coercion takes place when either of the operands is numeric.

In ECMA compliant JavaScript implementations, string values are simply compared according to the Unicode character code point values with no attempt to provide the more complex semantically oriented definitions of character and string equality defined in the Unicode version 2.0 specification.

The associativity is from left to right.

Refer to the operator precedence topic for details of execution order.

The result is the Boolean value `true` if *anOperand1* is numerically or lexically less than *anOperand2*, otherwise `false` is returned.

See also:	ASCII, Associativity, Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than or equal to (<=), Logical expression, Logical operator, NOT Equal to (!=), NOT Identically equal to (!==), Operator Precedence, Relational expression, Relational operator, Unicode

Cross-references:

ECMA 262 edition 2 –section –11.8.1

ECMA 262 edition 2 –section –11.8.5

ECMA 262 edition 3 –section –11.8.1

ECMA 262 edition 3 –section –11.8.5

Less than or equal to (<=) (Operator/relational)

Compare two operands to determine which is nearer to -Infinity or whether they are equal.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`anOperand1 <= anOperand2`*
Argument list:	*`anOperand1`*	A value that can be compared numerically or lexically
	`anOperand2`	A compatible value

Returns `true` if the left operand is numerically less than or equal to the right operand or is sorted earlier or identically in the Unicode collating sequence when two string values are compared.

In numeric comparisons, the presence of `NaN` in either or both operands will yield undefined instead of `true` or `false`.

When comparing two strings, a prefixing plus sign is present, then a numeric coercion of a string takes place before the comparison. Numeric coercion takes place when either of the operands is numeric.

In ECMA compliant JavaScript implementations, string values are simply compared according to the Unicode character code point values, with no attempt to provide the more complex semantically oriented definitions of character and string equality defined in the Unicode version 2.0 specification.

The associativity is from left to right.

Refer to the operator precedence topic for details of execution order.

The result is the Boolean value `true` if *`anOperand1`* is numerically or lexically less than or equal to *`anOperand2`*, otherwise `false` is returned.

See also:	ASCII, Associativity, Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Logical expression, Logical operator, NOT Equal to (!=), NOT Identically equal to (!==), Operator Precedence, Relational expression, Relational operator, Unicode

Cross-references:

ECMA 262 edition 2 –section –11.8.3

ECMA 262 edition 2 –section –11.8.5

ECMA 262 edition 3 –section –11.8.3

ECMA 262 edition 3 –section –11.8.5

Letter (Definition)

A valid and printable character.

Since JavaScript is available across such a wide variety of platforms, locales and environments, you may introduce some characters that appear to mutate as the script moves from one environment to another.

It is fairly safe to use any character in the lower 128 set that corresponds to ASCII. Even then, you may find that hash (#) and GB pound signs (£) get confused in some editors.

It is somewhat less safe to use the next 128 character codes up to 255 because these are mapped quite differently on Macintosh, UNIX and Windows systems.

If you can use a Unicode compliant environment, these problems may be minimized and the character you type will be the same on other platforms if they fully support Unicode. However they may not appear the same on non Unicode platforms. In any case, a non-Unicode compliant platform may not support the 16 bit (double-byte) characters that Unicode requires and may only be able to cope with 8 bit characters.

Historically, only 7 bit character codes could be guaranteed to arrive intact, but these days 7 bit serial connections are in decline. Even so, you may find them on some ancillary equipment; in particular you may find rack mounted units that have an RS232 serial interface. You should be very wary of these since they may not support 16 bit character values.

This set of characters (plus the space, newline and tab characters) is the smallest safe set to write your script in. Any other characters can be encoded as escape sequences using these characters.

a b c d e f g h i j k l m

n o p q r s t u v w x y z

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

! " # % & ' ( ) * + , –.

/ : ; < = > ? [`] ^ _ { | } ~

A fully ECMA compliant interpreter should allow you the full use of the Unicode character set for specifying identifier names but this whole area of localization and internationalization is under discussion at the time of writing. Future versions of the ECMA 262 standard or appendices to it may clarify the situation.

See also:	`isAlnum(),isAlpha(),isCtrl(),isDigit(),isGraph(),isLower(), isODigit(),isPrint(),isPunct(),isSpace(),isUpper(),isXDigit(),` Printing character

Lexical convention (Definition)

ECMAScript defined certain lexical conventions.

Availability:	ECMAScript edition –2

According to the ECMAScript standard, a compliant interpreter should follow a certain logical process when converting the source text of a script into an executable form. The standard defines a set of logical entities, which can be combined to make a program. This abstract definition is broad enough that all JavaScript interpreters can be considered to be ECMAScript compliant on this issue.

A script is composed of an unordered collection of the following kinds of entities:

- Whitespace
- Line terminators
- Comments
- Tokens

Whitespace is used to improve the readability of the script and to separate tokens from one another, where they could be misinterpreted if they were concatenated together.

Line terminators are used to improve the readability of the source text and to separate tokens. However, unlike whitespace, line terminators can affect the behavior of the script when placed in certain places. In general, a line terminator can occur between any two tokens but cannot appear in a token or inside a string literal. Line terminators in string literals must be escaped if they are required as part of the string. Line terminators also affect the automatic semicolon insertion process.

Comments can be contained completely on a single line or can span multiple lines.

A single line comment must finish at the end of the line and cannot contain a line terminator. You can place multiple single line comments one after another if required, each one separated by a line terminator.

Multiple line comments are replaced by a single line terminator during parsing. It doesn't matter how many line terminators they actually contain. This means that a multiple line comment behaves syntactically as if it were a line terminator.

Tokens are the actual components that your executable script is built from. They may be reserved words, identifiers, punctuator symbols or literals.

See also:	Automatic semicolon insertion, Comment, Comment (`//` and `/* ... */`), Escape sequence (`\`), Line terminator, Multi-line comment, Script Source Text, Token, Whitespace

Cross-references:

ECMA 262 edition 2 –section –7

ECMA 262 edition 3 –section –7

O'Reilly JavaScript Definitive Guide –page –27

O'Reilly JavaScript Definitive Guide –page –9-201

Lexical element (Overview)

A component from which a script is constructed.

A lexical element is a component from which you can construct a script. It is the smallest (or most atomic) fragment that a script can be decomposed into to allow the execution mechanisms in the interpreter to operate. Each lexical element is represented by a token and the script is tokenized through a process of lexical analysis.

The following lexical elements are used to construct a script:

- Whitespace
- Line terminator
- Literal (Constant)
- Identifier
- Keyword
- Reserved word
- Operator
- Punctuator
- Comment

See also:	Associativity, Comment, Constant, Identifier, `int`, JavaScript language, Keyword, Line terminator, Literal, Operator, Operator Precedence, Punctuator, Reserved word, Token, Whitespace

Lexical scoping (Definition)

The scope within which functions are executed.

Functions are lexically scoped in JavaScript and because they are not dynamically scoped, they are static to within the location they are defined. This means they run in the global scope of the document in which they live and not the one they are called from.

In versions of JavaScript prior to 1.2, the scope for a function declaration was limited to the global scope. This was a simple scope arrangement and caused little confusion and difficulty. From version 1.2 onwards, because functions can now be defined inside functions, they cannot be called from outside those functions and the scoping rules start to become more complex.

There becomes an issue of persistence of the scope of a function and if a nested function is called from outside its containing function, the containing function's scope is still present and added to the scope chain along the way when the inner function is called.

To illustrate this, suppose a function `AAA` is created and within it, another `BBB` is created.

You can call `AAA` and while executing its code, a call to `BBB` might be made. Inside `BBB`, the scope chain contains the `global` object, the `call` object of `AAA` and the `call` object of `BBB`. If `BBB` is called from outside `AAA`, as in `AAA.BBB()`, then the exact same scope chain is constructed.

This can become even more complex if `function` objects are manufactured at run-time. It might be possible to conceive a `function` object factory that can preserve the scope chain that persisted at the time the functions were created. If those `function` objects are preserved and executed later, the scope chain that was in existence at the time they were created will be restored when they are executed. This is far too confusing to be of great use and may turn out to be somewhat non-portable.

This capability is realized by storing the function that has been manufactured in a special kind of object. These are called `Closure` objects and are implemented visibly in Navigator 4.

See also:	`__parent__`, `Closure` object

LI object (Object/HTML)

An object that represents an `<LI>` object in the document.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0

Inherits from:	`Element object`	
JavaScript syntax:	IE	*myLI* = *myDocument*.all.*anElementID*
	IE	*myLI* = *myDocument*.all.tags("LI")[*anIndex*]
	IE	*myLI* = *myDocument*.all[*aName*]
	-	*myLI* = *myDocument*.getElementById(*anElementID*)
	-	*myLI* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myLI* = *myDocument*.getElementsByTagName("LI")[*anIndex*]
HTML syntax:	`<LI> ... </LI>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`type, value`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

The `<LI>` tag is a block-level tag. That means that it forces a line break before and after itself.

The DOM level 1 specification refers to this as a `LIElement` object

See also:	`Element` object, `OL.compact`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`type`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`value`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

LI.type (Property)

A type indicator that controls the presentation style of an item in the list that the <LI> object belongs to.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myLI*.type

There are a variety of different list types that can be displayed. The type value can be defined on an item by item basis within the ordered or unordered list collections.

Lists support an enumeration display of the following types:

<UL>/<OL>	Type code	Presentation style
OL	1	Numeric
OL	a	Alphabetical –lower case
OL	A	Alphabetical –upper case
OL	i	Roman numerals –lower case
OL	I	Roman numerals –upper case
UL	circle	a small open circular bullet
UL	disc	a small solid circular bullet
UL	square	a small square bullet

Some browsers may provide extended functionality, and use of characters other than those specified will yield undefined behavior. This is largely superseded by the CSS styling mechanisms now and is likely to become deprecated in due course.

See also:	OL.type, UL.type

LI.value (Property)

A means of resetting an enumerator in an ordered list contained in an <OL> block.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myLI*.value

You may want to reset an enumerator in a list sequence. This allows you to start the list at zero or one or perhaps to split the list and continue enumerating items after a body of text has been interposed into the list.

Of course this property is meaningless in an unordered list and is only therefore useful when the <LI> tag and LI object are contained within an <OL> tag and its corresponding OL object.

See also:	OL.start

Liberate TV Navigator (TV Set-top Box)

An interactive TV set-top box environment.

This is possibly the most advanced of the TV set top boxes that provide a 'Browser in a box'.

This system is gaining much popularity in the UK and Europe and is also becoming more prevalent in the United States. It has the backing of several very large players in the digital TV market and was founded as a joint venture between Netscape Communications and Oracle Inc.

Broadcast head end systems are generally built on Unix systems. Since the heritage of Liberate is that of the Netscape browser, the Liberate platform relates to the Microsoft TV platform in much the same way as the Netscape browser does to the MSIE browser.

From the consumer's point of view, one of the benefits of the Digital TV revolution is that generally the set-top boxes are supplied by the broadcasters. The broadcaster already has control over what gets transmitted from the head end and can now control how it is received. Although we have competing standards for set-top boxes, a TV set-top box will generally receive a predictable content model from the broadcast head end.

The Liberate platform does all that the WebTV and Microsoft TV platform does and possibly more besides, although both are competing with one another and raising the stakes all the time, just like it was during the 'Browser Wars'.

For example, you can request content from an ISP via a dial-up connection or from an always-on cable modem. This uses the normal HTTP request method. However, you can also request resources from the in-band broadcast video signal. These are done with additional request methods. These will allow you to request an item from a particular carousel. A carousel is a collection of assets that are transmitted in a cyclic manner and are embedded into the digital TV transport stream. Requesting an item from a carousel is no more complicated than requesting it from a web server. When the object is delivered on the next carousel cycle it is displayed by the browser in the normal way. Pages can be constructed with any mixture of in-band or online content.

Carousels can be associated with a particular channel or can be shared by many channels being broadcast on a digital TV multiplex.

This platform works very well in Digital Satellite (DSat), Digital Cable (DCable) and Digital Terrestrial (DTT) systems, although at this time it is becoming very popular on DCable.

The present version of Liberate in wide circulation is 1.1 and it is in the process of being upgraded to 1.2. The upgrade process is accomplished by downloading a new browser core into the set-top box which then updates its persistent Flash ROM.

Liberate 1.2 supports much enhanced carousel delivery, ATVEF triggers and also some basic Flash animation support. The JavaScript support is basically 1.2 and the HTML is based on version 3.2, which poses some difficulties in achieving the display effects people are used to deploying with HTML 4. These limitations are imposed because the set-top box hardware is based on a cost-limited CPU which has limited memory and performance. This is necessary to be able to deploy the boxes at a cost that is commercially viable.

As the cost/performance ratio improves, boxes will become more capable and support higher level functionality. It is likely that the Flash support for example will receive some considerable attention, because it allows the designers of the program related material to achieve more advanced and animated effects.

The JavaScript functionality has a few limiting factors when compared with the Netscape browser used in a PC. This extends to a few objects and methods being unavailable. However, there are additional TV-related objects and methods that provide the control of channel switching, volume control and TV overlays. You can for example use HTML to design a semi-transparent overlay that is placed on top of the video or you can place the window into a web page so it resembles an embedded video player plugin.

Major Cable operators and AOL TV are among the services offering contents that work with the Liberate platform.

See also:	ATVEF, Interpret, TV Set-top boxes

Light() (Filter/visual)

A visual filter for simulating a lighting model.

Availability:	JScript –3.0 Internet Explorer –4.0

See also:	filter – `Light()`

Limits (Definition)

A set of constraints within which the script must operate.

There are mainly two kinds of limits that an implementation imposes on a JavaScript script source text. There are those constraints required by the environment and there are numerical limits dictated by the internal data type representations.

Environmental limits are, for example, the security mechanisms in a web browser. Other environmental limits may impose a maximum limit to the size of a JavaScript source text. It is unusual to encounter this limit, but in an embedded interpreter the storage available to buffer the script as it is interpreted may be limited to 32K or 64K in situations where the script interpreter is used in a home appliance.

Numeric limits are those such as the size of the smallest and largest numeric value that can be represented or the maximum length of a string.

In general, the limits are either self-evident from the context in which the script is being executed or can be deduced by using certain properties of the `Global` object or the `Number` class.

See also:	Compliance, Environment, `Global` object, Minima-maxima, `Number` object

Line (Definition)

A fragment of script source text.

A line of script source text is that fragment of script that is placed between two consecutive line terminators.

The placement of comment blocks can affect where the line terminator appears to be.

Implementations may or may not strip off leading and trailing whitespace.

Trailing whitespace that is inside a string literal should not span a line terminator.

A line of script source is interpreted and executed as a whole.

Some implementations may place limits on the maximum length of a line of script source text. These limitations are most likely to be a problem in embedded interpreters and are least likely to cause any problems in web browsers.

See also:	Comment, Comment (// and /* ... */), Line terminator, Multi-line comment, Single line comment

Line terminator (Definition)

Line terminators separate individual lines of executable code.

Availability:	ECMAScript edition –2

Line terminators are used to improve the readability of the source text and to separate tokens. However, unlike whitespace, line terminators can affect the behavior of the script when placed in certain places.

In general, a line terminator can occur between any two tokens but cannot appear in a token or inside a string literal. Line terminators in string literals must be escaped if they are required as part of the string. Line terminators also affect the automatic semicolon insertion process.

Line terminators occurring during a single line comment delimited by a pair of slash characters (`//`) are considered to be the end of the comment, and any remaining comment text be interpreted as if it were executable code.

Line terminators occurring during a multiple line comment block (`/* ... */`) will be discarded and the entire comment block will be replaced with a single line terminator.

The following characters are considered to be line terminators in ECMAScript conformant JavaScript interpreters:

Escape Sequence	Unicode Value	Name	Symbol
\n	\u000A	Line Feed	<LF>
\r	\u000D	Carriage Return	<CR>
-	\2028	Line separator	<LS>
-	\2029	Paragraph separator	<PS>

The terms Line Terminator and Newline can for most purposes be used interchangeably. Newline is preferred when referring to a `\n` escape sequence since it more clearly suggests the meaning of the escape.

ECMA edition 3 adds the Unicode line separator and paragraph separator code points to the list of valid line terminators.

See also:	Automatic semicolon insertion, Comment (// and /* ... */), Lexical convention, Lexical element, Line, Newline, Semicolon (;), String literal

Cross-references:

ECMA 262 edition 2 –section –7.2

ECMA 262 edition 2 –section –7.8

ECMA 262 edition 3 –section –7.3

LINK object (Object/HTML)

An object that represents HTML <LINK> tags in documents.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myLINK* = *myDocument*.all.*anElementID*
	IE	*myLINK* = *myDocument*.all.tags("LINK")[*anIndex*]
	IE	*myLINK* = *myDocument*.all[*aName*]
	-	*myLINK* = *myDocument*.getElementById(*anElementID*)
	-	*myLINK* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myLINK* = *myLinkArray*[*anIndex*]
	-	*myLINK* = *myDocument*.getElementsByTagName("LINK")[*anIndex*]
HTML syntax:	<LINK>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	charset, disabled, href, hreflang, media, readyState, rel, rev, title, type	
Event handlers:	onClick, onDblClick, onError, onHelp, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReadyStateChange	

The <LINK> tag is used to link in external style sheet files. You can link in CSS or JSS style sheets with this technique. It allows the style sheets to be shared amongst many documents and for the site appearance to be changed globally simply by modifying a single file.

When referring to style sheets, the REL attribute has the STYLESHEET value.

The TYPE attribute indicates that the style sheet is formatted as text and contains JavaScript source text.

The REL and TYPE attributes combined tell us it is a JSS file.

The HREF attribute points at the document containing the style sheet definition to be loaded at the <LINK> point in the calling document.

This a LINK object because it refers to a document that is accessed via a URL.

The <LINK> tag conveys no apparent visible effect on the document. It is considered to be an invisible tag.

MSIE supports a LINK object as a property of its styleSheet object.

Warnings:

- This object is related to but not identical to a `Link` object. It does share some property names but adds others.
- This a special MSIE object class, although Netscape would probably support something functionally similar internally.

See also:	`Anchor` object, `Area` object, `Document.anchors[]`, `Element` object, `Element.all[]`, `LinkArray` object, `Location` object, `String.link()`, URL, `Url` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`charset`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`disabled`	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	-	-
`href`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`hreflang`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`media`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`readyState`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
`rel`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`rev`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`title`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`type`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onError`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onLoad`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onReadyStateChange`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

LINK.charset (Property)

The character set that the document at the other end of a link's URL is expected to use.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.charset`

This would contain the character set being used by the target document at the other end of the link. For example the value "`iso-8859-1`" is likely to be returned but the local variant of the browser and OS may affect the value you get.

This property might contain a value such as:

```
csISO5427Cyrillic
```

Details of other aliases can be located at the IANA registry.

See also:	`Anchor.charset,Url.charset`

Web-references:

ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

LINK.disabled (Property)

A switch property to enable or disable a link.

Availability:	DOM level –2 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myLINK*`.disabled`

Setting this property to `true` will disable a link from performing any action when clicked on by the user. Setting this property `false` restores its normal operation.

LINK.href (Property)

The URL of a document belonging to the `link` object (this is readonly in IE4 on the Macintosh).

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.href`

See also:	`Anchor.href`, `Location.href`, `Url.href`

LINK.hreflang (Property)

The language that the document at the other end of the URL is expected to conform to.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.hreflang`

This property should contain values that use the international language two-letter abbreviation codes. These are not the same as the country codes, which are also two letter values.

Refer to the Language codes topic for a list of the available language codes.

See also:	`Anchor.hreflang`, Language codes, `Url.hreflang`

LINK.media (Property)

The target media that the document will be output to.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.media`

This property is provided speculatively for a time when pages can be formatted differently according to the medium they are being delivered to.

One of the following values would be appropriate:

- `all`
- `print`
- `screen`
- `aural`
- `braille`
- `embossed`
- `handheld`
- `projection`
- `tty`
- `tv`

Warnings:

- Note that although this is supported by MSIE version 4, it does not work on the Macintosh implementation of that browser version.

LINK.readyState (Property)

The current disposition of the link as it is being loaded from the server.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myLINK*`.readyState`

This property reflects the downloading state of an object associated with a `<LINK>` tag.

Sometimes, you can design scripts to execute while the document is downloading; inline scripts for example. At that time, you may even be able to trigger interval-timed, deferred executions, as well.

If it is important that the document has completed loading before your script attempts to execute, you can check this property for one of the following values:

State	Value
uninitialized	The object is first instantiated but has not begun loading.
loading	The object has commenced loading.
loaded	The object has completed loading.
interactive	The object is loaded but not yet closed but is ready to handle interaction.
complete	The object body has been closed and the loading is finished.

An object may not need to reflect the complete status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an `OBJECT` object that represents an `<OBJECT>` tag until the `<BODY>` has completed loading. This is because the `ActiveX` object construction requires a complete document body structure to attach itself to.

Every time this `readyState` value changes, it triggers an `onReadyStateChange` event call-back.

See also:	`LINK.title`, `onReadyStateChange`

Property attributes:

`ReadOnly`.

LINK.rel (Property)

The relationship between the current element and the remote document.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.rel`
HTML syntax:	`<LINK REL="...">`	

This is sometimes called a forward link. Although the `HREF="..."` HTML tag attribute is normally the only means used to identify a target document, the browser is permitted to use the `REL="..."` HTML tag attribute to decide whether to use the `HREF` value or how it should be used.

The following HTML version 4.0 standard link types are permitted in this property:

- `alternate`
- `appendix`
- `bookmark`
- `chapter`
- `contents`
- `copyright`
- `glossary`
- `help`
- `index`
- `next`
- `prev`
- `section`
- `start`
- `stylesheet`
- `subsection`

MSIE adds these as well:

- `same`
- `next`
- `parent`
- `previous`

When used or tested within a script, any comparisons should be case-insensitive.

See also:	`Anchor.rel,Url.rel`

LINK.rev (Property)

The relationship between the remote document and the current element.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.rev`

This is sometimes called a reverse link. It defines the relationship between a document and another that calls it. The linkage is defined from the destination document's viewpoint.

This property supports the same HTML version 4.0 standard link types as the `rel` property. Refer to that topic for details.

When used or tested within a script, any comparisons should be case-insensitive.

As `rel` and `rev` properties are complementary, the values in them are likely to be related. For example, if one contains the value "`next`" then the other is likely to contain "`previous`".

See also:	`Anchor.rev`, `Url.rev`

LINK.title (Property)

The title name text of the link.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.title`

This is the text that is presented as a '`ToolTip`' when the mouse rolls onto an object and pauses there for a few moments. The MSIE browser will display this text as small popup comment box.

When aural style sheet support is provided by a browser, this value might be spoken as the mouse rolls over it.

See also:	`Document.title`, `Element.title`, `LINK.readyState`

LINK.type (Property)

The MIME type of the document that the link points at.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLINK*`.type`

The MIME type of the document associated with the LINK is accessible through the value of this property.

Refer to the MIME type topic for details of the available MIME types you will likely see in this property.

See also:	`Anchor.type`, MIME types, `Url.type`

LinkArray object (Object/browser)

A collection of `link` object, belonging to a document.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0	
JavaScript syntax:	-	*myLinkArray* = *myDocument*.links
Object properties:	`length`	

This is a collection of `Url` objects. In Netscape, you can inspect the constructor to establish the class name which is masked by the `toString()` method of the `Url` object. In MSIE, you cannot get at the constructor so we have to assume that the object is a `Url` object.

`Url` objects are created when an `<AREA>` or `<A>` tag refers to a document. Anchors that are simply named locations within a document but which don't have an `HREF` get added to the anchors array but not to the links array.

Netscape prior to version 6.0 calls this a `LinkArray` (as opposed to a `LinksArray` which might be more appropriate). In MSIE it is just a `Collection` and in Netscape version 6.0 it has become an `HTMLCollection` because that is what DOM specifies it should be.

Warnings:

- Be careful not confuse the elements of this array with `LINK` objects. These are used in MSIE to support styling of `Url` objects on the screen. Other documentation may refer to `Link` objects but there is no evidence to support the existence of an object of that class. After inspection there appear to be `Url` objects in Netscape, LINK objects in MSIE and an object in MSIE that corresponds to the Netscape `Url` class but which provides no means of examining its constructor.

See also:	`Area` object, `Collection` object, `Document.anchors[]`, `Document.links[]`, `HyperLink` object, `LINK` object, `Url` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`length`	1.0 +	3.0 +	2.0 +	4.0 +	-	-	ReadOnly.

LinkArray.length (Property)

The number of HREF links in the current document.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myDocument*.links.length

The length of the links array, which indicates the number of <A> and <AREA> tags that contain HREF attributes in the document.

See also:	Collection.length, Document.links[], Url.name

Property attributes:

ReadOnly.

LinkStyle object (Object/DOM)

Added at DOM level 2 to support linked stylesheets.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
JavaScript syntax:	N	*myLinkStyle* = new LinkStyle()

DOM level 2 specifies that this object should support the following property:

- sheet

List type (Definition)

An internal type used by the interpreter.

Availability:	ECMAScript edition –2

This is an internal type used by the interpreter for processing expression evaluation results. It cannot be stored as an object property.

Although the List type does not really exist as a data type accessible from a script, the internal behavior of the interpreter appears as if it did. Especially when looking at the way argument lists are processed in new operator expressions.

See also:	`new`, Type

Cross-references:

ECMA 262 edition 2 –section –11.2.4

ECMA 262 edition 2 –section –8.8

ECMA 262 edition 3 –section –8.8

ECMA 262 edition 3 –section –11.2.4

LISTING object (Object/HTML)

An object that represents the `<LISTING>` tag.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myLISTING* = *myDocument*.all.*anElementID*
	IE	*myLISTING* = *myDocument*.all.tags("LISTING")[*anIndex*]
	IE	*myLISTING* = *myDocument*.all[*aName*]
	-	*myLISTING* = *myDocument*.getElementById(*anElementID*)
	-	*myLISTING* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myLISTING* = *myDocument*.getElementsByTagName ("LISTING")[*anIndex*]
HTML syntax:	`<LISTING>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

The `LISTING` object is instantiated when the browser encounters a `<LISTING>` tag in the HTML for a document. This tag is used to enclose a section of text that should be presented as if it were a computer code listing.

The appearance will be similar to that rendered by a <CODE>, <PRE> or <KBD> tag.

Use of this tag is highly deprecated but it still persists in some legacy content.

See also:	Element object, KBD object, PRE object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Node object

Literal (Definition)

Constant values used to initialize or assign a value.

Literals are constant values used in assignments or as arguments to functions or expressions. A literal is considered to be a primary expression when it is being evaluated.

There are several kinds of literals. They are all based on the primitive types. The ECMA standard defines four basic data types, which should be sufficient for most purposes; however, a hosted interpreter may provide other more esoteric primitive values. Here are the four core standard types:

- Null
- Boolean
- Numeric
- String

These are objects that can also be manufactured from literals:

- Array
- Date
- Regular expression
- Object
- Void

String literals also include escaped character literals which may also be used in regular expressions.

Host environments additionally define others for special purposes depending on the implementation.

See also:	Boolean literal, Character constant, Constant, Lexical element, Null literal, Numeric literal, Primary expression, String literal, Token

Cross-references:

ECMA 262 edition 2 –section –7.7

ECMA 262 edition 2 –section –11.1.3

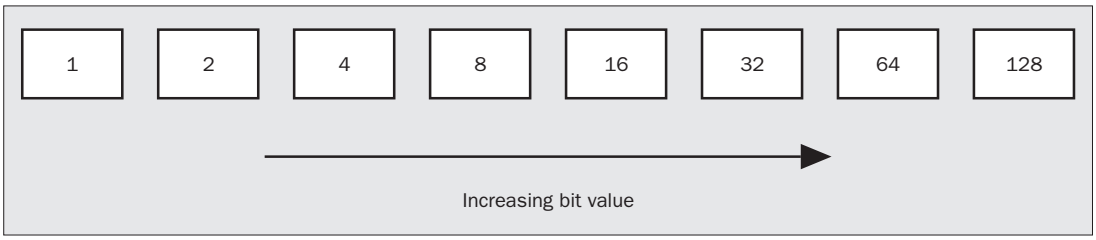
ECMA 262 edition 3 –section –7.8

ECMA 262 edition 3 –section –11.1.3

O'Reilly JavaScript Definitive Guide –page –30

Little endian (Definition)

A bit ordering standard for some CPU models.



See also:	byte

LiveConnect (Product)

A Netscape Communications technology that provides a means of communicating between plugins and Java applets.

In Netscape, communication between scripts and Java applets takes place via the LiveConnect interface. LiveConnect was originally designed for just this purpose. LiveConnect is only implemented in Netscape. However, similar but not identical functionality is available in MSIE via the ActiveX interface.

LiveConnect is most effective if you can also understand the internals of Java applets and can program in Java yourself. You can use LiveConnect to gain read and write access to the public fields of an applet and you can also invoke public methods. You can also interact with standard Java system classes that are built into the Netscape browser.

If a plugin is Java-enabled, then in Netscape you can interact with that plugin in the same way.

From the opposite direction, applets and Java-enabled plugins can invoke JavaScript functions and can read and write JavaScript object properties. The Real Video plugin uses this to great effect as do many other media plugins.

LiveConnect is constructed around some built-in objects that encapsulate various Java classes and objects. These are:

- `JavaObject` –An encapsulation of an instance of a Java object that belongs to a `JavaClass`
- `JavaClass` –An encapsulation of a Java class for access to static properties and methods
- `JavaPackage` –A collection of `JavaClass` and `JavaPackage` objects in a tree-like hierarchy

With these encapsulations, you can operate on Java objects as if they were JavaScript objects. That is necessary because although you can generally convert between primitive types, the object types are too dissimilar to convert properly.

There are some interesting opportunities here for extending the capabilities of your scripts with the facilities offered by Java. However, the downside is that you need to have a VM running. Netscape supports its own internal VM at least up to version 4, but the trend is for browsers to use the OS hosted VM and to not provide any embedded Java execution context. This may provide additional benefits such as sharing Java classes amongst several applications and shortening the start-up time since the VM could be expected to be running already.

LiveConnect becomes particularly useful when accessing the `java.lang.System` class from scripts running in Netscape. In MSIE, you could create a special applet that then gained access to the system class for you. Using this capability to any useful extent starts to impinge on the security model and requires that trusted scripts are used. This is usually enough to deter most people from exploring it more deeply. However, if you can gain that trust, by using signed scripts, you may then be able to access the Java IO facilities and read/write files.

Working the other way, LiveConnect provides a way for Java code to execute a string as JavaScript. This is very like an `eval()` call from the native JavaScript environment.

Warnings:

- Be careful not to confuse the following:
  - LiveWire
  - LiveScript
  - LiveConnect
- In Netscape version 3, a bug in the conversion routines yielded all Java primitives as JavaScript objects instead of JavaScript primitives. This causes a few problems because `Number` and `Boolean` objects in JavaScript behave slightly differently to the primitive representations, mostly to do with operator overloading and precedence. Objects will prefer to concatenate as strings rather than add as numbers for example. The `valueOf()` method will provide a way to work around this.

See also:	ActiveX, Glue code, Java calling JavaScript, Java method calls, Java method data conversion, Java to JavaScript values, JavaScript to Java values, LiveScript, Plugin events, `Plugin.isActive()`

LiveScript (Product)

This was the original name for JavaScript when it was first introduced by Netscape Communications.

Maybe LiveScript wouldn't have caught on and become so popular, but now people routinely confuse it with Java. Even more so now that interpreters for JavaScript are available written in Java and some server-side application tools allow you to write Java code but run it in a script interpreter. That would be Java script not JavaScript.

Warnings:

- Be careful not to confuse the following:
  - LiveScript
  - LiveWire
  - LiveConnect
- LiveWire is the former name for Server-Side Javascript.
- LiveConnect is a mechanism for allowing JavaScript to talk to Java in a Netscape browser.

See also:	JavaScript language, JavaScript version, LiveConnect, LiveWire

livescript: URL (Request method)

This is a pseudonym for the `javascript:` URL.

See also:	URL, `javascript:` URL

LiveWire (Product)

Netscape Communications' server-side JavaScript compiler.

This was the original name for JavaScript when used in the server. Now it is simply called Server-Side JavaScript. Sometimes SSJS for short.

The LiveWire JavaScript interpreter is used in Netscape Enterprise Server. In this environment you can also compile the JavaScript code which further blurs the distinction between Java and JavaScript.

Warnings:

- Be careful not to confuse the following:
  - LiveWire
  - LiveScript
  - LiveConnect

See also:	LiveScript, Server-side JavaScript

Local time (Definition)

The locale specific time value.

Availability:	ECMAScript edition –2

Local time is the calendar time for the current locale.

Local time and UTC time differ by an amount derived by adding the Local Time Zone Adjustment and the Daylight Savings Time Adjustment together. Adding these to the UTC time gives the local time.

Conversion of UTC time (`ut`)to local time is defined by:

```
LocalTime(t) = ut + LocalTZA + DaylightSavingTA(ut)
```

Conversion from local time (`lt`)to UTC is defined by:

```
UTC(lt) = lt - LocalTZA - DaylightSavingTA(lt - LocalTZA)
```

The following expression may not always test `true`:

```
t = UTC(LocalTime(t))
```

Converting from UTC to local time and back may not yield an identical value due to deficiencies in daylight savings time computations. ECMA does not mandate that a compliant implementation needs to take DST into account only that it be aware that DST may be in force. Some locales use the same DST setting all year round in some years and that is difficult to predict and encapsulate into an algorithm.

See also:	Broken down time, Calendar time, Date and time, Daylight savings time adjustment, Local time zone adjustment

Cross-references:

ECMA 262 edition 2 –section –15.9.1.9

ECMA 262 edition 3 –section –15.9.1.9

Local time zone adjustment (Definition)

An adjustment to locale specific time.

Availability:	ECMAScript edition –2

ECMA compliant implementations are expected to determine the local time zone adjustment.

The local time zone adjustment is a value (internally referred to as LocalTZA) which is measured in milliseconds and can be added to the UTC value representing the local standard time.

The ECMA standard states that daylight savings time is not reflected in the LocalTZA value. LocalTZA does not vary with time but depends only on the geographic location.

See also:	Broken down time, Daylight savings time adjustment, Local time

Cross-references:

ECMA 262 edition 2 –section –15.9.1.7

ECMA 262 edition 3 –section –15.9.1.7

Locale-specific behavior (Definition)

Behavior that depends on a locale setting.

The locale setting is a way of defining how programs operate in an international or geographic context. Running software in London and New York may be very similar apart from the instantaneous time setting. The time in New York is 5 hours behind London. If the software running in both locations needs to communicate with the other and will have to perform time based operations on data, then to remain properly synchronized they may need to be aware of the number of time zones between the two systems and make adjustments accordingly.

Other behavior may dictate date formats, currency symbols, the format of decimal values, thousands separators and spelling. Non English speaking locales may use the same textual presentation (Roman Scripting) but spell words in the local language. Character set substitution may accomplish much of what's required although this is somewhat ameliorated by the use of Unicode. Sometimes the direction that the text flows may be different. For example, European, Middle-Eastern and Asian languages use different character alphabets and text directions.

The implementors should thoroughly document all of this locale specific behavior.

See also:	Behavior, Broken down time, Character set, Character-case mapping, Unicode

Localization (Definition)

The process by which geographic, international and regional differences are managed.

JavaScript is used all over the world in a wide variety of environments, platforms and contexts. Portability issues arise when moving between platforms. They are also present when relocating a platform geographically. At the very least a time setting will need to change, and to record an event in a coordinated time framework, the offset from that known time reference needs to be known. There will be cultural, language and currency changes necessary as well as number formatting rule variations that need to be taken account of.

Indeed the differences in some locales may completely change the native alphabet that the user is prepared to work with. Even the direction of the text flow may change in extreme circumstances. Moving around Europe certainly requires that different accented symbols be used in each country.

Fortunately the Unicode standard covers a great many of these national variants.

Even so, JavaScript is fundamentally based on the American English language and even with the international support of the Unicode character set, the structural layout of the script source text may still conform to the underlying English language form. Even so there are issues with the spelling of words such as Color.

Typically, a localized implementation may allow the use of accented and national characters in identifier names but would still retain the English spelled keywords and punctuation characters.

As well as character sets, the issue of collation into the correct sorting sequence needs to be managed properly. In JavaScript, this issue is largely confined to the way that `Array` objects manage their `sort()` method's behavior.

Date and time formats tend to be arranged differently in each country. The United States and the United Kingdom commonly place the day number and month number the opposite way round. For day numbers above 12, this can be detected automatically but otherwise the context needs to be examined to decide which way round these values are.

Numeric formats may dictate a different punctuation symbol to be used for the decimal point and thousands separator. Confusingly, the comma and period characters may be used in both contexts. This means that a value could be misinterpreted and might yield a result that is several orders of magnitude in error when parsed.

Currency symbols will need some attention. The United States dollars symbol is in the lower 128 character set. The cents symbol is not. The UK pounds symbol is within the lowest 255 characters.

Warnings:

- Be careful of the following:
  - Date formats
  - Time settings
  - Character sets
  - Currency symbols
  - Sort ordering
  - Decimal points and thousands separators
- The entire concept and scope of localization of JavaScript is being discussed as part of the standardization project. This area is prone to change or, at the very least, some definition of a standard conforming behavior.

See also:	Character set, Collation sequence, Currency symbol, Date and time, Decimal point (.), Multi-byte character

location (Property)

An alias for the `window.location` property.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Location object`	
JavaScript syntax:	-	`location`
	-	*`myWindow`*`.location`

See also:	`Window.navigate(),Window.location`

Property attributes:

`ReadOnly.`

Location object (Object/DOM)

An object that represents the location of a document.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0
Inherits from:	`Url object`
JavaScript syntax:	- *myLocation* = *myWindow*.location
Object properties:	`hash, host, hostname, href, pathname, port, protocol, search, target, text, x, y`
Object methods:	`assign(), reload(), replace()`
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`

This is a more or less portable encapsulation of a URL value and is most commonly used to describe the location of a document in a window. Changing its `href` property value has become the preferred means of loading a new document into a window or frame.

This is useful because it can also operate very conveniently across frame boundaries. That is, a script in one frame can modify the contents of another.

There are some security implications when accessing frames from different servers or domains. These can overcome in Netscape by using signed scripts.

See also:	`Anchor` object, `Area` object, `Document.location`, `LINK` object, `Map` object, URL, `Url` object, `Window.location`, `Window.navigate()`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`hash`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`host`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`hostname`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`href`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`pathname`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`port`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`protocol`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	ReadOnly.
`search`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`target`	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	-
`text`	1.2 +	-	4.0 +	-	-	-	-	-
`x`	1.2 +	-	4.0 +	-	-	-	-	-
`y`	1.2 +	-	4.0 +	-	-	-	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`assign()`	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	Warning, Deprecated
`reload()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	-
`replace()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object, `Url` object

Location.assign() (Method)

Equivalent to setting the `HREF` attribute to load a new page.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Deprecated	
JavaScript syntax:	-	*myLocation*`.assign(`*aURL*`)`
Argument list:	*aURL*	A new URL to load into the `location.href`

This is functionally equivalent to making an assignment with an equals (=) operator.

Warnings:

- This method was not originally intended for use by script level code and should be avoided.

See also:	Assign value (=)

Location.hash (Property)

The hash target portion of the `href` property. (The hash after the URL refers to an anchor location in the document).

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLocation*.hash
See also:	`Anchor.hash,Url.hash`	

Location.host (Property)

The hostname and port number portion of the `href` property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLocation*.host
See also:	`Anchor.host,Url.host`	

Location.hostname (Property)

The hostname-only portion of the `href` property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLocation*.hostname
See also:	`Anchor.hostname,Url.hostname`	

Location.href (Property)

The URL for the page currently on display in the window owning this location.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLocation*.href

If you set this property, the window will load the new URL in, and replace the old content with the new.

See also:	`Anchor.href,Url.href`

Location.pathname (Property)

The file path portion of the `href` property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLocation*.pathname

See also:	`Anchor.pathname,Url.pathname`

Location.port (Property)

The port number within the `href` property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0

Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myLocation*.port

See also:	`Anchor.port`, `Url.port`

Location.protocol (Property)

The protocol portion of the `href` property, such as: `http:`,`ftp:`,`mailto:`,`file:`,etc.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myLocation*.protocol

See also:	`Anchor.protocol`, `IMG.protocol`, URL, `Url.protocol`

Property attributes:

`ReadOnly`.

Location.reload() (Method)

Reload the currently displayed page in the window that owns this `location` object.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
JavaScript syntax:	-	*myLocation*.reload()
	-	*myLocation*.reload(*aFlag*)
Argument list:	*aFlag*	A Boolean flag forcing a reload from the server when set to `true`

This causes the document that owns the `Location` object to be reloaded. Any form elements would be reset to their initial values.

Unless you specify an unconditional reload, the page will be reloaded from the cache if the file is still in the cache, and caching is still active. This may involve a check on the server to see if the copy in the cache is up to date. If the cache copy is out of date, a new document may be fetched from the server.

You can use the keyword unconditional as an argument in the `location.reload(true)` method call to force a new copy of the document to be fetched from the server regardless of the disposition of any documents in the local cache. This should also defeat any proxies and force them to reload from the server but it depends on how they are configured.

Refer to the `History.go()` topic for details of how to perform a benign soft load.

See also:	`History.go()`

Location.replace() (Method)

Load a new page and replace the history entry.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
JavaScript syntax:	-	*myLocation*`.replace(`*aURL*`)`
Argument list:	*aURL*	A URL to load into the layer, window or frame this location belongs to

This method is a useful way of loading new pages if you want to make sure the history list does not get filled up with unwanted pages. This makes the back button more useful to the user.

While the replace goes on, the existing page will continue to be displayed. This also applies if the replace is called during the `<HEAD>` portion of a page. The new URL is requested but the browser will continue loading and building the previous page until the web server responds and delivers the new replacement page. Termination of the old page therefore happens when the new page begins to arrive and not when it is first requested.

Location.search (Property)

The query portion of the `href` property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myLocation*.search

See also:	Anchor.search, Url.search

Location.target (Property)

The target window or frame that a location belongs to when its object represents an <A> tag.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myLocation*.target

This yields the value of the TARGET attribute in an <A>, <AREA> or <MAP> tag.

You can assign a new value to this property so that the URL will be directed to a different window or frame.

Here are some example target values:

- _parent
- _self
- _top
- _blank
- Window name
- Frame name

See also:	Anchor.target, BASE.target, Form.target, Url.target

Location.text (Property)

The text contained within the <A> and </A> tags when the location describes an anchor.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myLocation*.text

See also:	Anchor.text, Url.text

Location.x (Property)

The X coordinate of the anchor on the client display surface.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLocation*.x

The horizontal position of the object in the display measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

See also:	`Anchor.x`, `Area.x`, `Url.x`

Location.y (Property)

The Y coordinate of the anchor on the client display surface.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myLocation*.y

The vertical position of the object in the display measured in pixels. You can use the x and y coordinates of the object as targets of the `scrollTo()` method for the window it lives in.

See also:	`Anchor.y`, `Area.y`, `Url.y`

locationbar (Property)

An alias for the `window.locationbar` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Bar object`	
JavaScript syntax:	-	locationbar
	-	*myWindow*.locationbar

See also:	`Bar` object, `Window.locationbar`

Property attributes:

`ReadOnly.`

Lock object (Object/NES)

Provides a way of locking objects against multiple simultaneous access by several clients at once.

Availability:	JavaScript –1.2 Netscape Enterprise Server version –3.0	
JavaScript syntax:	NES	*myLock* = Lock
	NES	*myLock* = new Lock()
Object properties:	constructor, prototype	
Object methods:	isValid(), lock(), unlock()	

In a server back end environment, you may have some objects which you share amongst several sessions. This means that they could be accessed on behalf of several clients all at once. You may want to prevent this happening by locking the object as the first client's request accesses it, and then unlocking it again as the request handler for that client no longer needs access to it.

You can create new `lock` objects with the `Lock()` constructor.

It is good manners to unlock resources as soon as you can so that other processes can carry on running. You should not rely on the script exit handler unlocking the locks you place on objects.

Warnings:

- Be careful when you use locks. It is possible to place mutually exclusive locks on objects and introduce a deadlocking situation. There is a potential for this if your script is locking more than one object and the locks are not correctly nested.
- If you don't implement locks when necessary, you run the risk of run-time errors as your system can run out of resources.

See also:	Netscape Enterprise Server, `unwatch()`, `watch()`

Property	JavaScript	JScript	NES	Notes
constructor	1.2 +	-	3.0 +	-
prototype	1.2 +	-	3.0 +	-

Method	JavaScript	JScript	NES	Notes
isValid()	1.2 +	-	3.0 +	-
lock()	1.2 +	-	3.0 +	-
unlock()	1.2 +	-	3.0 +	-

Lock() (Constructor)

A constructor for creating new `Lock` objects.

Availability:	JavaScript –1.2 Netscape Enterprise Server version –3.0	
Property/method value type:	`Lock` object	
JavaScript syntax:	NES	`new Lock();`

See also:	`Lock` object

Lock.constructor (Property)

A reference to the constructor for making new `Lock` objects.

Availability:	JavaScript –1.2 Netscape Enterprise Server version –3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	NES	*`myLock`*`.constructor`

The constructor is that of the built-in `Lock` prototype object.

You can use this as one way of creating locks although it is more popular to use the `new Lock()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Lock.isValid() (Method)

A method that returns an indication as to the validity of the lock.

Availability:	JavaScript –1.2 Netscape Enterprise Server version –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	*`myLock`*`.isValid()`

This method will return a Boolean `true` value if the `Lock` is still in force and a Boolean `false` value if it has been relinquished.

See also:	`Lock.lock(),Lock.unlock()`

Lock.lock() (Method)

Place a lock on the object.

Availability:	JavaScript –1.2 Netscape Enterprise Server version –3.0	
JavaScript syntax:	NES	*myLock*.lock()

As you attempt to lock the object, the method first tests the `Lock` to see if another script already has a lock pending. If it does, then this method stalls and will not return until it can successfully place a lock on the object. That will happen when the previous locking script calls the `unlock()` method.

See also:	`Lock.isValid()`

Lock.prototype (Property)

The prototype for the `Lock` object that can be used to extend the interface for all `Lock` objects, by allowing you to add methods and properties.

Availability:	JavaScript –1.2 Netscape Enterprise Server version –3.0	
Property/method value type:	`Lock object`	
JavaScript syntax:	NES	`Lock.prototype`
	NES	*myLock*.constructor.prototype

See also:	`prototype` property

Lock.unlock() (Method)

Relinquish a lock on the object.

Availability:	JavaScript –1.2 Netscape Enterprise Server version –3.0	
JavaScript syntax:	NES	*myLock*.unlock()

You should always aim to relinquish a lock as soon as you can. They are not intended for use as a means of reserving server facilities but as a means of preventing clashes between sessions. If the unlocking is successful, then `true` will be returned, and `false` otherwise.

See also:	`Lock.isValid()`

Logical AND (&&) (Operator/logical)

Logical AND of two operands.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*anOperand1* `&&` *anOperand2*
Argument list:	*anOperand1*	A Boolean value
	anOperand2	Another Boolean value

Traditionally in programming environments, the logical AND operator yields `true` only when both operands are true. However, the specifics of this are slightly different in JavaScript, and although the results may appear to be functionally the same, there is a subtle but important difference.

First, lets deal with the normal and expected behavior of a Logical AND operator. The truth table shows the result of this operator for two Boolean primitive values:

A	B	AND
false	false	false
false	true	false
true	false	false
true	true	true

Now, the implementation is expected to conform to the ECMA standard. This sets out the following method of evaluation for a Logical AND operator:

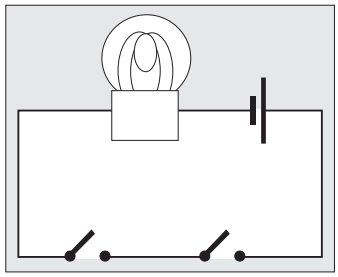
- Evaluate and convert the first operand using the `ToBoolean()` method.
- If it is `false`, then evaluate and return the second operand.
- Otherwise return the evaluation of the first operand.

To all intents and purposes the external perceived behavior is the same because another `ToBoolean()` conversion is likely to take place in the context that the expression is used –an `if()` statement or an outer logical expression for example.

The associativity is from left to right.

Refer to the operator precedence topic for details of execution order.

The result is the Boolean value `true` if both operands are true, otherwise Boolean `false` is returned.



Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=1 CELLPADDING=2>
<TR>
<TH>A</TH>
<TH>B</TH>
<TH>AND</TH>
</TR>
<SCRIPT>
for(a=0; a<2; a++)
{
   for(b=0; b<2; b++)
   {
      document.write("<TR ALIGN=CENTER><TD>");
      document.write(Boolean(a));
      document.write("</TD><TD>");
      document.write(Boolean(b));
      document.write("</TD><TD>");
      document.write(Boolean(a && b));
      document.write("</TD></TR>");
   }
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	Associativity, Binary logical operator, Bitwise AND (&), Bitwise AND then assign (&=), Logical expression, Logical operator, Operator Precedence

Cross-references:

ECMA 262 edition 2 –section –11.11

ECMA 262 edition 3 –section –11.11

Logical constant (Definition)

A Boolean constant value.

There are only two possible values for a Boolean constant. That is `true` or `false`.

A Boolean variable may contain the value undefined.

See also:	Constant expression

Logical entity (Definition)

Boolean logical values are represented as an entity internally.

Availability:	ECMAScript edition –2

Logical entities always evaluate as a Boolean result.

Logical entities are used to form conditions and control branching among other uses. They are often used to hold state information or control switches. A logical entity may be the result of a relational expression or the use of one or more logical operators. A logical entity may be the result of enquiring the property of an object that returns a Boolean value as a result. Logical entities can only contain a `true` or `false` value.

Cross-references:

ECMA 262 edition 2 –section –4.3.14

ECMA 262 edition 3 –section –4.3.14

Logical expression (Definition)

An expression whose result is a Boolean value.

Availability:	ECMAScript edition –2
Property/method value type:	`Boolean primitive`

Logical operators perform a test of the Boolean value of the two operands either side of the operator.

See also:	Equal to (==), Expression, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), Logical AND (&&), NOT Equal to (!=), NOT Identically equal to (!==)

Cross-references:

ECMA 262 edition 2 –section –11.11

ECMA 262 edition 3 –section –11.11

Logical NOT – complement (!) (Operator/logical)

Logical NOT operator.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`!` *`anOperand`*
Argument list:	*`anOperand`*	A Boolean value to be complemented

The result is the Boolean complement of the operand value.

The exclamation mark is the logical negation operator. The operand is evaluated and its result converted to a binary value. The value is then reversed and used to replace the expression in whatever context it has been used.

The truth table shows the result of this operator for a Boolean primitive value:

A	NOT
false	true
true	false

Although this is classified as a logical operator here, it is also classified as a unary operator since it only has one operand.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=1 CELLPADDING=2>
<TR>
<TH>A</TH>
<TH>NOT</TH>
</TR>
<SCRIPT>
for(a=0; a<2; a++)
{
```

```
    document.write("<TR ALIGN=CENTER><TD>");
    document.write(Boolean(a));
    document.write("</TD><TD>");
    document.write(Boolean(!a));
    document.write("</TD></TR>");
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	Associativity, Bitwise NOT –complement (~), Logical operator, NOT Equal to (! =), Operator Precedence, Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 –section –11.4.9

ECMA 262 edition 3 –section –11.4.9

Logical operator (Definition)

An operator that creates a logical (Boolean) expression.

Availability:	ECMAScript edition –2
Property/method value type:	`Boolean primitive`

Logical operators perform a one bit operation on the Boolean value of the operands. The input values may not be Boolean values but will be converted to a Boolean value using the `toBoolean` rules for that type. The conversion happens automatically but may not always be what you expect.

The JavaScript interpreter is also quite smart, in that it can for some operator and value combinations perform a lazy evaluation. For example if an expression like this is presented:

A AND B

Then if A is false, B does not need to be evaluated to know that the result will be `false`. This can significantly speed up the interpreter because evaluating B might have involved an object reference or perhaps an expression evaluation. It almost always would involve a type conversion from some other type to a Boolean value. This suggests you may effect some performance gains by coding your logical operations to take advantage of this.

There are specific logical operators in the set for performing logical AND as well as logical OR operations. However, generally speaking any operator that yields a Boolean result could be used in the same place. The following table summarizes the logical operators as well as those, which come under that general heading of providing a Boolean result:

Operator	Description
`&&`	Logical AND
`||`	Logical OR
`!`	Logical NOT
`==`	Equal to
`!=`	NOT equal to
`<`	Less than
`<=`	Less than or equal to
`>`	Greater than
`>=`	Greater than or equal to

The Boolean constants `true` and `false` need to be considered too, because they are often used in conditional tests to force the conditions to be `true` or `false`. Although they are not operators as such they can be useful when constructing expressions for debugging purposes.

Type conversions indicate these relationships:

- `false` becomes 0
- `true` becomes 1
- 0 becomes `false`
- Any non-zero value becomes `true`

Note that this is going to lose some information if you convert a non-zero value other than 1 to a Boolean value and back again.

Warnings:

- This is not to be confused with the bitwise operators, which yield a 32-bit integer value instead of the Boolean value yielded by a logical expression.
- The bitwise operators may yield a value that in other languages is the same as the logical operator. However although in C language, `true` and `false` are really integer values, in JavaScript the Boolean and Number values are distinctly different types.
- Be careful to use the correct number of ampersands, and vertical bars to select the logical version of the operator. Refer to the Bitwise operator topic for a list of operators to avoid in logical expressions.

See also:	Associativity, Binary logical operator, Bitwise operator, Equal to (==), Expression, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), Logical AND (&&), Logical NOT –complement (!), Logical OR (||), NOT Equal to (!=), NOT Identically equal to (!==), Operator, Operator Precedence, `ToBoolean`, Type, Type conversion

Cross-references:

ECMA 262 edition 2 –section –11.11

ECMA 262 edition 3 –section –11.11

Wrox *Instant JavaScript* –page –19

Logical OR (||) (Operator/logical)

Logical OR of two operands.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*anOperand1* `||` *anOperand2*
Argument list:	*anOperand1*	A Boolean value
	anOperand2	Another Boolean value

Traditionally in programming environments, the logical OR operator yields `true` when either or both of the operands are `true`. It yields `false` only when both are `false`. However, the specifics of this are slightly different in JavaScript and although the results may appear to be functionally the same, there is a subtle but important difference.

First, lets deal with the normal and expected behavior of a Logical OR operator. The truth table shows the result of this operator for two Boolean primitive values:

A	B	OR
false	false	false
false	true	true
true	false	true
true	true	true

Now, the implementation is expected to conform to the ECMA standard. This sets out the following method of evaluation for a Logical OR operator:

- Evaluate and convert the first operand using the `ToBoolean()` method.
- If it is `true`, then return that operand.
- Otherwise evaluate and return the second operand.

To all intents and purposes the external perceived behavior is the same because another `ToBoolean()` conversion is likely to take place in the context that the expression is used –an `if()` statement or an outer logical expression for example.

The associativity is from left to right.

Refer to the operator precedence topic for details of execution order.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=1 CELLPADDING=2>
<TR>
<TH>A</TH>
<TH>B</TH>
<TH>OR</TH>
</TR>
<SCRIPT>
for(a=0; a<2; a++)
{
   for(b=0; b<2; b++)
   {
      document.write("<TR ALIGN=CENTER><TD>");
      document.write(Boolean(a));
      document.write("</TD><TD>");
      document.write(Boolean(b));
      document.write("</TD><TD>");
      document.write(Boolean(a || b));
      document.write("</TD></TR>");
   }
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	Associativity, Binary logical operator, Logical operator, Operator Precedence

Cross-references:

ECMA 262 edition 2 –section –11.11

ECMA 262 edition 3 –section –11.11

Logical XOR (Operator/logical)

Logically exclusive OR of two values.

There isn't really an XOR logical operator, but the Bitwise XOR operator should work fine.

This is proven by evaluating the truth table in the example.

This seems to work fine even on MSIE where the sign bit exhibits some instability under Bitwise operations.

This is the truth table for two Boolean primitive values being operated on with the XOR operator.

A	B	XOR
false	false	false
false	true	true
true	false	true
true	true	false

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=1 CELLPADDING=2>
<TR>
<TH>A</TH>
<TH>B</TH>
<TH>XOR</TH>
</TR>
<SCRIPT>
for(a=0; a<2; a++)
{
   for(b=0; b<2; b++)
   {
      document.write("<TR ALIGN=CENTER><TD>");
      document.write(Boolean(a));
      document.write("</TD><TD>");
      document.write(Boolean(b));
      document.write("</TD><TD>");
      document.write(Boolean(a ^ b));
      document.write("</TD></TR>");
   }
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	Bitwise XOR (^)

long (Reserved word)

Reserved for future language enhancements.

The inclusion of this reserved keyword in the ECMAScript standard suggests that future versions of ECMAScript may be more strongly typed.

This keyword also represents a Java data type and the `long` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	`double`, `float`, `Integer`, `java.lang.Long`, LiveConnect, Reserved word, `short`

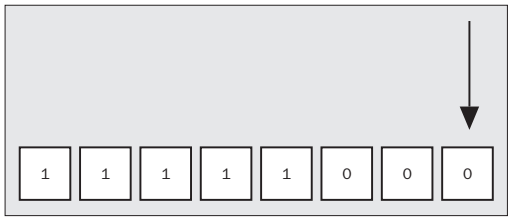
Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Low order bit (Definition)

The least significant bit in an integer value.



See also:	Bit, Bit-field, Bitwise operator, byte

LValue (Definition)

`LValues` are placed on the left of an assignment.

Availability:	ECMAScript edition –2

An `LValue` is that expression or identifier reference that can be placed on the left hand side of an assignment.

An `LValue` must be a modifiable entity.

These are called Left-Hand-Side expressions in the ECMA standard.

The C language refers to these as Locator Values.

See also:	= (Assign), Add then assign (+=), Assign value (=), Assignment expression, Bitwise AND then assign (&=), Bitwise OR then assign (|=), Bitwise shift left then assign (<<=), Bitwise shift right and assign (>>=), Bitwise unsigned shift right and assign (>>>=), Bitwise XOR and assign (^=), Concatenate then assign (+=), Conversion, Divide then assign (/=), Left-Hand-Side expression, Multiply then assign (*=), Property value, Remainder then assign (%=), RValue

Cross-references:

ECMA 262 edition 2 –section –11.2

ECMA 262 edition 3 –section –11.2

mailbox: URL (Request method)

Displays the Netscape Message Center window.

You can add keywords to access the individual components within the mailbox as necessary. The following are all valid URL values in Netscape:

- mailbox:
- mailbox:Inbox
- mailbox:Unsent%20Messages
- mailbox:Drafts
- mailbox:Templates
- mailbox:Sent
- mailbox:Trash

Each of these will spawn an additional window as required to display the relevant folder within the mailbox. Note that embedded spaces must be URL escaped for them to work. This mechanism does not seem to provide access to sub-folders.

Warnings:

- Although Netscape 6.0 supports this request method, its support is incomplete as of the first non-beta release.

See also:	javascript: URL, mailto: URL, nethelp: URL, telnet: URL, URL

mailto: URL (Request method)

Activates the mail client to send an e-mail message.

Open up a mail client application or use the browser's built-in mail client to send a message to the indicated recipient.

Netscape supports the predefiniton of the `Subject:`,`Cc:` and `Bcc:` fields in the `mailto:` URL. This is illustrated in the example. This may not work on other browsers or JavaScript-capable mail clients.

Note that you can leave the spaces as they don't need to be escaped in the Subject text. In some versions, you can also define the Body text for the e-mail by appending the `Body` operator. This is also shown in the example.

Warnings:

- This is only allowed under script control if the script has the `UniversalSendMail` privilege.

Example code:

```
<!-- Simple example showing just a mail to link -->
<a href="mailto:someone@somewhere.com">Send Mail</a>
<!-- Adding a subject header -->
<a href="mailto:someone@somewhere.com?Subject=Email From
Link&Cc=access@wrox.com&Bcc=backup@wrox.com">Send Mail</a>
<!-- Adding a subject header -->
<a href="mailto:someone@somewhere.com?Subject=Email From Link">Send Mail</a>
<!-- Adding CC and BCC fields -->
<a href="mailto:someone@somewhere.com?Subject=Email From Link">Send Mail</a>
<!-- Adding a predefined message body -->
<a href="mailto:someone@somewhere.com?Subject=Email From
Link&Cc=access@wrox.com&Bcc=backup@wrox.com&Body=messageText">Send Mail</a>
```

See also:	javascript: URL, mailbox: URL, nethelp: URL, UniversalSendMail, URL

main() function (Definition)

The main entry point to a procedural language program.

Property/method value type:	Implementation defined

Although JavaScript in many circumstances does not have a `main()` function, in some applications of the language it may be necessary.

By implication, the `main()` function of a JavaScript is the global code that is present in a web page in the `<SCRIPT>` tags, but is not part of a function declaration body.

When embedded in a browser, the language is invoked according to an event model and the browser itself is the `main()` for the interpreter. Various script based functions are called by the browser.

In a server-side environment, a script may be used as the result of a CGI call. In that case, a `main()` with passed parameters is a more useful facility.

In some implementations, the `main()` function may be used to make the language accessible to C programmers. Languages often borrow from one another and JavaScript borrows heavily from C language and Java. The exact form of the `main()` function and possibly its name will be implementation dependent and you should check the documentation for the interpreter just make sure it behaves the way you expect.

The result of calling the `main()` function will be implementation specific, but probably an integer.

See also:	`argc` parameter, `argv` parameter, Execution context, Execution environment, Host environment, `Host` object

MakeDate() (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

The `MakeDate()` operator calculates a number of milliseconds from its two arguments. Both arguments must be ECMAScript number values.

All of the arguments must be numeric and finite values. The values should be integers. If they are not then the result will be `NaN`.

The day argument is multiplied by the milliseconds per day and the time argument is added to the result. The sum of the two is a millisecond coded date and time value.

Although this is called an operator in the standard, its behavior is more like that of a function. It is not part of the formal language implementation but is a useful function to have available, and can be simulated by writing a script-based function. It is documented in the standard to assist in the algorithmic breakdown of the `Date` method handlers.

The result is a date value in milliseconds since the base date.

See also:	`Cast` operator, `Date` object, `Date()`, `Date.UTC()`, Time range, Time value

Cross-references:

ECMA 262 edition 2 –section –15.9.1.13

ECMA 262 edition 3 –section –15.9.1.13

MakeDay() (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

The `MakeDay()` operator calculates a number of days from its three arguments. Each argument must be an ECMAScript number value.

All of the arguments must be numeric and finite values. The values should be integers. If they are not then the result will be `NaN`.

The month number is used as a lookup to establish the number of days in the preceding months. This will also take the year number into account because a leap year calculation may be necessary.

The standard does not mandate any range checking, however it would be sensible to include range checks in the implementation of this functionality.

Although this is called an operator in the standard, its behavior is more like that of a function. It is not part of the formal language implementation, but is a useful function to have available and can be simulated by writing a script-based function. It is documented in the standard to assist in the algorithmic breakdown of the `Date` method handlers.

The result is a day value in milliseconds since the base date.

See also:	`Cast` operator, `Date` object, `Date()`, `Date.UTC()`, Time range, Time value

Cross-references:

ECMA 262 edition 2 –section –15.9.1.12

ECMA 262 edition 3 –section –15.9.1.12

MakeTime() (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

The `MakeTime()` operator calculates a number of milliseconds from its four arguments. Each argument must be an ECMAScript number value.

All of the arguments must be numeric and finite values. The values should be integers. If they are not then the result will be `NaN`.

The constant time values giving milliseconds per hour, minute and second are used to multiply each argument. The results are then summed to provide an equivalent time in milliseconds.

No range checking is performed according to the ECMA standard so some degree of overflow can be expected.

Although this is called an operator in the standard, its behavior is more like that of a function. It is not part of the formal language implementation, but is a useful function to have available and can be simulated by writing a script-based function. It is documented in the standard to assist in the algorithmic breakdown of the `Date` method handlers.

The result is a time value since midnight measured in milliseconds.

See also:	`Cast` operator, `Date` object, `Date()`, `Date.UTC()`, Time range, Time value

Cross-references:

ECMA 262 edition 2 –section –15.9.1.11

ECMA 262 edition 3 –section –15.9.1.11

<MAP TARGET="..."> (HTML Tag Attribute)

The frame or window to target by default with the links in an image map.

This is a non-standard tag attribute that is used as a means of linking frames and area maps. You should use `<AREA TARGET="...">` instead of `<MAP TARGET="...">` for portable applications.

Warnings:

- Because this is a non-standard HTML tag attribute, most implementations will not support the `TARGET` attribute on a `<MAP>` tag. In that case, apply the `TARGET` attribute to the `<AREA>` tags in the map instead.

See also:	`Anchor.target`, `Form.target`, `Map.target`, `Window.frames[]`

Deprecated usage:

Yes

Map object (Object/HTML)

An object that represents a `<MAP>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myMap* = *myDocument*.all.*anElementID*
	IE	*myMap* = *myDocument*.all.tags("MAP")[*anIndex*]
	IE	*myMap* = *myDocument*.all[*aName*]
	-	*myMap* = *myDocument*.getElementById(*anElementID*)
	-	*myMap* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myMap* = *myDocument*.getElementsByTagName("MAP")[*anIndex*]
HTML syntax:	`<MAP> ... </MAP>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`name, target`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	
Collections:	`areas[]`	

The `<MAP>` tag in the HTML document source is a container for the `<AREA>` tags. These all belong to a parent `Map` object. This is a means of componentizing a client-side image map and building complex non-rectangular shaped areas.

See also:	`A` object, `Area` object, `Element` object, `Location` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`name`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`target`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5+	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

Map.areas[] (Collection)

A collection of `Area` objects belonging to the `map` object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	-	*myMap*.areas

The objects instantiated by `<AREA>` tags which are part of this image map, are collected together and accessible as a collection. The collection object is referred to by this property.

Property attributes:

`ReadOnly`

Map.name (Property)

The value of the `NAME="..."` HTML tag attribute.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myMap*.name

Objects are identified either by the `NAME="..."` HTML tag attribute or by the `ID="..."` HTML tag attribute.

Netscape Navigator shows a marginal preference for the `name` property while MSIE seems slightly better disposed towards the `ID` property. However, in many cases both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

See also:	`IMG.useMap`

Map.target (Property)

The target window or frame to which a map applied.

Availability:	JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myMap*.target
HTML syntax:	`<MAP TARGET="...">`	

This yields the value of the `TARGET` attribute in an `<A>`, `<AREA>` or `<MAP>` tag.

You can assign a new value to this property so that the URL will be directed to a different window or frame.

Here are some example target values:

- `_parent`
- `_self`
- `_top`
- `_blank`
- Window name
- Frame name

See also:	`<MAP TARGET="...">`,`Anchor.target`,`BASE.target`, `Form.target`,`Url.target`

MARQUEE object (Object/HTML)

An object that represents a `<MARQUEE>` HTML tag.

Availability:	JScript –3.0 Internet Explorer –4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myMARQUEE* = *myDocument*.all.*anElementID*
	IE	*myMARQUEE* = *myDocument*.all.tags("MARQUEE")[*anIndex*]
	IE	*myMARQUEE* = *myDocument*.all[*aName*]
	-	*myMARQUEE* = *myDocument*.getElementById(*anElementID*)
	-	*myMARQUEE* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myMARQUEE* = *myDocument*.getElementsByTagName("MARQUEE")[*anIndex*]
HTML syntax:	`<MARQUEE>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`accessKey, behavior, bgColor, dataFld, dataFormatAs, dataSrc, direction, height, hspace, loop, scrollAmount, scrollDelay, tabIndex, trueSpeed, vspace, width`	
Object methods:	`start(), stop()`	
Event handlers:	`onAfterUpdate, onBlur, onBounce, onClick, onDblClick, onDragStart, onFilterChange, onFinish, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onScroll, onSelectStart, onStart`	

The `MARQUEE` object is only supported by MSIE and provides means of quickly and easily generating a moving ticker display inside the window area. This can be accomplished with Netscape using layers and some interval timed JavaScript function calls to scroll the layer.

The MSIE `MARQUEE` object is a little more aware of Font Metrics than your average script access can achieve. This means making the `MARQUEE` bounce back and forth when the text hits the edge of the extent rectangle is a lot easier to accomplish in MSIE than in Netscape.

See also:	`Element` object, `Input.accessKey`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`accessKey`	-	3.0 +	-	4.0 +	-	-	-	-
`behaviour`	-	3.0 +	-	4.0 +	-	-	-	-
`bgColor`	-	3.0 +	-	4.0 +	-	-	-	-
`dataFld`	-	3.0 +	-	4.0 +	-	-	-	-
`dataFormatAs`	-	3.0 +	-	4.0 +	-	-	-	-
`dataSrc`	-	3.0 +	-	4.0 +	-	-	-	-
`direction`	-	3.0 +	-	4.0 +	-	-	-	-
`height`	-	3.0 +	-	4.0 +	-	-	-	-
`hspace`	-	3.0 +	-	4.0 +	-	-	-	-
`loop`	-	3.0 +	-	4.0 +	-	-	-	-
`scrollAmount`	-	3.0 +	-	4.0 +	-	-	-	-
`scrollDelay`	-	3.0 +	-	4.0 +	-	-	-	-
`tabIndex`	-	3.0 +	-	4.0 +	-	-	-	-
`trueSpeed`	-	3.0 +	-	4.0 +	-	-	-	-
`vspace`	-	3.0 +	-	4.0 +	-	-	-	-
`width`	-	3.0 +	-	4.0 +	-	-	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`start()`	-	3.0 +	-	4.0 +	-	-	-	-
`stop()`	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onBounce`	-	3.0 +	-	4.0 +	-	-	-	-
`onClick`	1.0 +	3.0 +	4.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFinish	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	3.0 +	4.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-
onStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

MARQUEE.behaviour (Property)

A attribute that controls the motion of the text in the `MARQUEE` object.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myMARQUEE*`.behavior`

The animated effect in the `MARQUEE` is controlled by this property. The following values are appropriate:

- `alternate`
- `scroll`
- `slide`

The alternate behavior marches left and then on bouncing into a boundary, marches right until it bounces again.

The scroll and slide behaviors use the direction attribute to control the direction of movement.

When set to scroll, the MARQUEE starts empty, the text scrolls in, traverses the MARQUEE and scrolls out again. A complete cycle begins and ends with an empty MARQUEE.

When set to slide, the MARQUEE again starts empty, the text scrolls in until it hits a boundary. Then it is simply zapped out and does not scroll away. Again, a complete cycle begins and ends with an empty MARQUEE.

MARQUEE.bgColor (Property)

The background color for the area occupied by the MARQUEE object.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myMARQUEE*.bgColor

The background can be colored independently of whether an image is loaded into the background of an object. In fact, it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	color names, color value

MARQUEE.direction (Property)

The direction of movement of the scrolling text in the MARQUEE.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myMARQUEE*.direction

A MARQUEE object can scroll in all four cardinal directions. Leftward scrolling is most likely the appropriate setting for textual content written in a Roman script that is read from left to right. Diagonal scrolling is not possible with a MARQUEE but could be implemented using layers in Netscape 4 or an IFRAME in MSIE and Netscape 6.0.

The following values are appropriate for this property:

- left
- right
- down
- up

MARQUEE.height (Property)

The height of the area allocated to the `MARQUEE` object as it appears onscreen.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myMARQUEE`*`.height`

The `MARQUEE` space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

MARQUEE.hspace (Property)

The height of the area allocated to the `MARQUEE` object as it appears onscreen.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myMARQUEE`*`.hspace`

Margins placed around objects are either modified separately with all four margin sides having a different property or by adjusting the horizontal margins and vertical margins using just two values.

The `hspace` property controls the margin to the left and right of the object.

MARQUEE.loop (Property)

A count of the number of times the `MARQUEE` text is to scroll.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myMARQUEE`*`.loop`

You can define the number of cycles the `MARQUEE` animation will cycle round before stopping. When it stops, the text will remain visible.

Continuous looping is specified by setting this property to a value of -1.

MARQUEE.scrollAmount (Property)

The offset of the text between one scroll cycle and the next.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myMARQUEE*`.scrollAmount`

This value is specified in pixels, and controls the distance that the text appears to move on each scroll cycle of the `MARQUEE`. The larger the number, the faster the scrolling appears to travel.

MARQUEE.scrollDelay (Property)

The time-delay in milliseconds between each scroll update.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myMARQUEE*`.scrollDelay`

A short delay and small scroll distance will consume more CPU time but appear to be a smoother scroll.

MARQUEE.start() (Method)

A command method to start the `MARQUEE` scrolling.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myMARQUEE*`.start()`

You can stop and start the `MARQUEE` animation whenever you want. Maybe it is appropriate to scroll a `MARQUEE` when a mouse is positioned over an object. You can call this method as part of a `MouseOver` event handler. Then, you can call its complementary `MARQUEE.stop()` method on the `MoveMouseOut` event handler.

MARQUEE.stop() (Method)

A command method to stop the `MARQUEE` from scrolling.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myMARQUEE*`.stop()`

Refer to:

`MARQUEE.start()`

MARQUEE.trueSpeed (Property)

A switch attribute that controls whether the browser should honor very small scroll delay times.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myMARQUEE*`.trueSpeed`

Extremely small delay times may cause the `MARQUEE` to scroll more than once during each screen retrace interval. It is wasteful of the CPU resources to scroll the `MARQUEE` faster than the viewer can perceive the changes. A longer delay and greater scroll distance is more efficient although it may appear jerky for extreme settings.

MARQUEE.vspace (Property)

The size of a vertical margin above and below the `MARQUEE` with respect to any adjacent objects.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myMARQUEE*`.vspace`

Margins placed around objects are either modified separately with all four margin sides having a different property or by adjusting the horizontal margins and vertical margins using just two values.

The `vspace` property controls the margin at the top and bottom of the object.

MARQUEE.width (Property)

The width of the area allocated to the `MARQUEE` object as it appears onscreen.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myMARQUEE*`.width`

The `MARQUEE` space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

Mask() (Filter/visual)

A visual filter for creating a transparent mask.

Availability:	JScript –3.0 Internet Explorer –4.0

Refer to:

Filter – `Mask()`

MaskFilter() (Filter/visual)

Uses the transparent color pixels of an object as a mask.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `MaskFilter()`

Math object (Object/core)

A globally available object containing a library of mathematical functions.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
JavaScript syntax:	-	`Math`
Object properties:	`constructor`	
Class constants:	`E, LN10, LN2, LOG10E, LOG2E, PI, SQRT1_2, SQRT2`	
Functions:	`abs(), acos(), asin(), atan(), atan2(), ceil(), cos(), exp(), floor(), log(), max(), min(), pow(), random(), round(), sin(), sqrt(), tan()`	

The `Math` object is merely a single object owned by the `Global` Object and which cannot be instantiated. It has some named properties, some of which are functions while others are constants.

The prototype for the `Math` prototype object is the `Object` prototype object.

Although it cannot be instantiated, it does have a constructor which in turn has a prototype property. By adding functions to that prototype, you can extend the capabilities of the `Math` object. Several examples are provided in nearby topics to illustrate the addition of extra trigonometric and hyperbolic functions.

Warnings:

- The `Math` object provides a collection of static constant values by way of properties belonging to the integral `Math` object. Because the mathematical mechanisms of any application tend to be provided by the operating system, you should find that between different browsers on any particular platform, the values that these constants yield will be very consistent.
- The ECMA standard lays down strict values for these properties and in general, the browser manufacturers try to comply –however, there is always the possibility that an implementation may use a non-compliant calculation.
- However, it may not be quite so reliable across platforms. You might enumerate one of these constants as you are authoring and then hard code that value into your script. When that script is executed on another platform, even in the same browser, the internal mathematics support may yield a different value.
- You should always refer to the static constants using their symbolic names rather than hard code a possibly platform-dependent value into your script.
- Note for the trigonometric functions in general that certain implied identities cannot be assumed in JavaScript. For example:
  - `Math.sin(Math.PI/2)` may not yield exactly 1
  - `Math.cos(Math.PI)` may not return precisely zero
  - `Math.acos(0)` may not return the same value as Math.PI
  - `Math.SQRT1_2` may not be exactly equal to the reciprocal of `Math.SQRT2`

See also:	Constant, Exponent-log function, `Global` object, Integer arithmetic, Java to JavaScript values, `Native` object, Object constant, `Object` object, Range error, Trigonometric function, Type conversion, `unwatch()`, `watch()`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`constructor`	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	-

Cross-references:

ECMA 262 edition 2 –section –10.1.5

ECMA 262 edition 2 –section –15.1.4.1

ECMA 262 edition 2 –section –15.8

ECMA 262 edition 3 –section –10.1.5

ECMA 262 edition 3 –section –15.1.5.1

ECMA 262 edition 3 –section –15.8

Math.abs() (Function)

The absolute value of a positive or negative number.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.abs(`*`aValue`*`)`
Argument list:	*`aValue`*	Some meaningful numeric value

This function returns the absolute value of the argument.

The absolute value of a number is its distance from zero.

In general, the result has the same magnitude as the argument but always has a positive sign.

Special boundary conditions that affect the results are:

Argument	Result
0	0
NaN	NaN
negative infinity	positive infinity

On some implementations, the absolute value of the most negative integer number may not be representable in the positive range.

This is not the same as `Number.MIN_VALUE` and `Number.MAX_VALUE`. They describe the largest and smallest possible positive floating point values.

Warnings:

- It is possible that due to the underlying implementation of the math library, the absolute value of the most negative number may not be representable and it may yield `NaN` instead.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
for(myEnum = 1.5; myEnum > -2; myEnum -= 0.1)
```

```
{
   document.write(myEnum + " " + Math.abs(myEnum) + "<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Integer arithmetic, Integer-value-remainder, `Math` object, `Math.ceil()`,`Math.floor()`, Type conversion

Cross-references:

ECMA 262 edition 2 –section –15.8.2.1

ECMA 262 edition 3 –section –15.8.2.1

Math.acos() (Function)

The inverse cosine of the passed-in value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.acos(`*`aValue`*`)`
Argument list:	*`aValue`*	Some meaningful numeric value

This function returns the arc-cosine of the argument.

Special boundary conditions that affect the results are:

Argument	Result
1	0
greater than 1	NaN
less than -1	NaN
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

The result of this method is the arc-cosine of the passed-in value. The result is expressed in radians and ranges from 0 to pi.

Warnings:

- Note that `Math.acos(0)` may not return the same value as `Math.PI`.

See also:	`Math` object, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.PI`, `Math.sin()`, `Math.tan()`, Trigonometric function

Cross-references:

ECMA 262 edition 2 –section –15.8.2.2

ECMA 262 edition 3 –section –15.8.2.2

Math.asin() (Function)

The inverse sine of the passed-in value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.asin(`*`aValue`*`)`
Argument list:	*`aValue`*	Some meaningful numeric value

This function returns the arc-sine of the argument.

Special boundary conditions that affect the results are:

Argument	Result
0	0
greater than 1	NaN
less than -1	NaN
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

The result of this method is the arc-sine of the passed-in value. The result is expressed in radians and ranges from -pi/2 to +pi/2.

See also:	`Math` object, `Math.acos()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`, `Math.tan()`, Trigonometric function

Cross-references:

ECMA 262 edition 2 –section –15.8.2.3

ECMA 262 edition 3 –section –15.8.2.3

Math.atan() (Function)

The inverse tangent of the passed-in value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.atan(`*`aValue`*`)`
Argument list:	*`aValue`*	Some meaningful numeric value

This function returns the arc-tangent of the argument.

Special boundary conditions that affect the results are:

Argument	Result
0	0
minus infinity	-pi/2
NaN	NaN
plus infinity	+pi/2

The exact value yielded by this function may vary slightly from implementation to implementation due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

The result of this method is the arc-tangent of the passed-in value. The result is expressed in radians and ranges from -pi/2 to +pi/2.

See also:	`Math` object, `Math.acos()`, `Math.asin()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`, `Math.tan()`, Trigonometric function

Cross-references:

ECMA 262 edition 2 –section –15.8.2.4

ECMA 262 edition 3 –section –15.8.2.4

Math.atan2() (Function)

The inverse tangent of the slope of the two arguments.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.atan2(`*`aValue1, aValue2`*`)`
Argument list:	*aValue1*	Some meaningful numeric value
	aValue2	Some meaningful numeric value

This function computes the quotient of arg2/arg1 and then returns the arc-tangent of the result. It takes into account which quadrant the value falls into according to the signs of the two arguments.

It is provided in several languages other than JavaScript for those situations where very large numbers are involved.

Traditionally, for this function, the Y argument is placed first and the X argument second.

Special boundary conditions that affect the results are:

Argument1	Argument2	Result
+infinity	+infinity	+pi/4
+infinity	-infinity	+3pi/4
+infinity	non zero	+pi/2
-infinity	+infinity	-pi/4
-infinity	-infinity	-3pi/4
-infinity	non zero	-pi/2
0	negative	pi * sign of Argument1
0	positive	0
< 0	-infinity	-pi
< 0	0	-pi/2
> 0	-infinity	+pi
> 0	0	+pi/2
Any value	NaN	NaN
NaN	Any value	NaN
non zero	+infinity	0

The exact value yielded by this function may vary slightly from implementation to implementation due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

The result of this method is the arc-tangent of the slope of the two arguments. The result is expressed in radians and ranges from -pi to +pi.

See also:	`Math` object, `Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.cos()`, `Math.sin()`, `Math.tan()`, Trigonometric function

Cross-references:

ECMA 262 edition 2 –section –15.8.2.5

ECMA 262 edition 3 –section –15.8.2.5

Math.ceil() (Function)

The value rounded up to the next higher integer value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.ceil(`*`aValue`*`)`
Argument list:	*`aValue`*	A meaningful numeric value

According to the ECMA standard, this function returns the smallest number value that is not less than the argument and is equal to a mathematical integer. If the argument is already an integer, the argument itself is returned.

Special boundary conditions that affect the results are:

Argument	Result
+infinity	+infinity
-1 < argument > 0	0
-infinity	-infinity
0	0
NaN	NaN

Note that if the value is negative, the magnitude decreases while it increases for positive numbers. The `ceil` of 25.4 is 26 whereas the `ceil` of -25.4 is -25.

The result is the input value rounded up to the next higher integer value.

Warnings:

- Other reference sources on this function differ as to its functionality. Some indicate that it rounds up to an integer, others that it rounds down. This suggests that some implementations may behave differently since they may use the available documentation to design their interpreter. The ECMA standard is probably the most reliable specification long term since manufacturers will attempt to build ECMA compliance into their implementations as a selling point.
- Recent browsers from Microsoft, Netscape and Opera are all compliant with the ECMA standard on this point.
- If you are uncertain, you should check the functionality by running some tests. For example, run the test for negative and positive values. Checking a few boundary conditions won't hurt either.

See also:	Integer, Integer arithmetic, Integer-value-remainder, `Math` object, `Math.abs()`,`Math.floor()`,`Math.round()`,`Number()`, `Remainder (%)`, Remainder then assign (%=), Type conversion

Cross-references:

ECMA 262 edition 2 –section –15.8.2.6

ECMA 262 edition 3 –section –15.8.2.6

Math.constructor (Property)

A means of creating a new `Math` object.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Math object`	
JavaScript syntax:	-	`Math.constructor`

Both Netscape and MSIE support a constructor property for the `Math` object. It is unlikely you would ever want to construct a copy of the `Math` object.

However, this is useful not for the purpose of cloning the `Math` object but so that you can extend it by adding your own properties and methods to its prototype.

This technique is demonstrated in the example which appears to add a `pythag()` function to the `Math` object although it is really added to the `Object()` object prototype.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that extends the capabilities of the Math object
function pythag(aValue1, aValue2)
{
   return Math.sqrt((aValue1*aValue1) + (aValue2*aValue2));
}
// Register the new function
Math.constructor.prototype.pythag = pythag;
// Test the Math.pythag() method
document.write(Math.pythag(3, 4))
document.write("<BR>")
</SCRIPT>
</BODY>
</HTML>
```

Math.cos() (Function)

The cosine of the input argument.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.cos(`*`aValue`*`)`
Argument list:	*`aValue`*	An angle measured in radians

This function returns the cosine of the argument. The argument must be expressed in radians.

Special boundary conditions that affect the results are:

Argument	Result
+infinity	NaN
-infinity	NaN
0	1
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation due to differences in the underlying precision of the implementations math routines and the specific algorithm selected to evaluate this function.

Warnings:

- Note that `Math.cos(Math.PI)` may not return precisely zero.

See also:	`Math` object, `Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.PI`, `Math.sin()`, `Math.tan()`, Trigonometric function

Cross-references:

ECMA 262 edition 2 –section –15.8.2.7

ECMA 262 edition 3 –section –15.8.2.7

Math.cosec() (Simulated functionality)

The `cosec()` function is not available in JavaScript but can be simulated to aid in the porting of existing code.

The example demonstrates how to add the `cosec()` method to the `Math` object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Add the cosec function to the Math object
function cosec(aValue)
{
   return 1/Math.sin(aValue);
}

// Register the new function
Math.constructor.prototype.cosec = cosec;

// Test the Math.cosec() method
document.write(Math.cosec(2));
document.write("<BR>")
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Math.cot()`, `Math.sec()`

Math.cosh() (Simulated functionality)

The `cosh()` function is not available in JavaScript but can be simulated to aid in the porting of existing code.

The example demonstrates how to add the `cosh()` method to the `Math` object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Add the cosh function to the Math object
function cosh(aValue)
{
   var myTerm1 = Math.pow(Math.E, aValue);
   var myTerm2 = Math.pow(Math.E, -aValue);
   return (myTerm1+myTerm2)/2;
}

// Register the new function
Math.constructor.prototype.cosh = cosh;

// Test the Math.cosh() method
document.write(Math.cosh(2));
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

Math.cot() (Simulated functionality)

The `cot()` function is not available in JavaScript but can be simulated to aid in the porting of existing code.

The example demonstrates how to add the `cot()` method to the `Math` object to provide the capability to calculate cotangents.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Add the cot function to the Math object
function cot(aValue)
{
   return 1/Math.tan(aValue);
}
```

```
// Register the new function
Math.constructor.prototype.cot = cot;

// Test the Math.cot() method
document.write(Math.cot(2));
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Math.cosec()`,`Math.sec()`

Math.E (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape Navigator –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.E`

The numeric constant value for e, the base of natural logarithms.

The resulting value is approximately 2.718281828459045 (to 15 d.p.).

Warnings:

- The word approximately is used when describing the result, because the mathematical accuracy of JavaScript implementations leaves something to be desired and there are some strange artifacts in some of the calculations.

See also:	Arithmetic constant, Exponent-log function, Floating point constant, `Math` object, `Math.exp()`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.1

ECMA 262 edition 3 –section –15.8.1.1

Math.exp() (Function)

The exponential function of the passed-in argument.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.exp(`*`aValue`*`)`
Argument list:	*`aValue`*	A meaningful numeric value

This function returns the exponential function of the argument (e raised to the power of the argument, where e is the base of the natural logarithms).

Special boundary conditions that affect the results are:

Argument	Result
+infinity	+infinity
-infinity	0
0	1
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation, due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

See also:	Exponent-log function, `Math` object, `Math.E`, `Math.LN10`, `Math.LN2`, `Math.log()`, `Math.LOG10E`, `Math.LOG2E`

Cross-references:

ECMA 262 edition 2 –section –15.8.2.8

ECMA 262 edition 3 –section –15.8.2.8

Math.floor() (Function)

The value is rounded down to the next integer.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape Navigator –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.floor(`*`aValue`*`)`
Argument list:	*`aValue`*	A meaningful numeric value

Returns the greatest number value that is not greater than the argument and is equal to a mathematical integer. If the argument is already an integer, the argument itself is returned.

Special boundary conditions that affect the results are:

Argument	Result
+infinity	+infinity
-infinity	-infinity
0	0
0 < argument > 1	0
NaN	NaN

Note that if the value is negative, the magnitude increases while it decreases for positive numbers. The floor of 25.4 is 25 whereas the floor of -25.4 is -26.

The result is the input value rounded down to the next integer.

Warnings:

- Other reference sources on this function differ as to its functionality. Some indicate that it rounds up to an integer, others that it rounds down. However, all implementations appear to conform to the ECMA specified behavior.

See also:	Integer, Integer arithmetic, Integer-value-remainder, `Math` object, `Math.abs()`,`Math.ceil()`,`Math.round()`,`Number()`, Remainder (%), Remainder then assign (%=), Type conversion

Cross-references:

ECMA 262 edition 2 –section –15.8.2.9

ECMA 262 edition 3 –section –15.8.2.9

Math.LN10 (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- `Math.LN10`

This constant provides the numeric value for the natural logarithm of 10.

The resulting value value is approximately 2.302585092994046 (to 15 d.p.)

Warnings:

- The word approximately is used when describing the result, because the mathematical accuracy of JavaScript implementations leaves something to be desired and there are some strange artifacts in some of the calculations.

See also:	Arithmetic constant, Exponent-log function, Floating point constant, `Math` object, `Math.exp()`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.2

ECMA 262 edition 3 –section –15.8.1.2

Math.LN2 (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0

Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.LN2`

This constant provides the numeric value of the natural logarithm of 2.

The resulting value returned is approximately 0.693147180559945 (to 15 d.p.)

See also:	Arithmetic constant, Exponent-log function, Floating point constant, `Math` object, `Math.exp()`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.3

ECMA 262 edition 3 –section –15.8.1.3

Math.log() (Function)

The natural logarithm of the passed-in value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.log(`*`aValue`*`)`
Argument list:	*`aValue`*	A meaningful numeric value

This function returns the natural (base e) logarithm of the input argument.

This function is the inverse of the `Math.exp()` function.

Special boundary conditions that affect the results are:

Argument	**Result**
+infinity	+infinity
0	-infinity
1	0
< 0	NaN
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation, due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

See also:	Exponent-log function, `Math` object, `Math.exp()`

Cross-references:

ECMA 262 edition 2 –section –15.8.2.10

ECMA 262 edition 3 –section –15.8.2.10

Math.LOG10E (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera browser –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.LOG10E`

This constant provides the numeric value for the base-10 logarithm of e, the base of natural logarithms.

Note that the value of `Math.LOG10E` is approximately the reciprocal of the value of `Math.LN10`.

The result returned is approximately 0.434294481903252 (to 15 d.p.)

Warnings:

- The word approximately is used when describing the result, because the mathematical accuracy of JavaScript implementations leaves something to be desired and there are some strange artifacts in some of the calculations.

See also:	Arithmetic constant, Exponent-log function, Floating point constant, `Math` object, `Math.exp()`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.5

ECMA 262 edition 3 –section –15.8.1.5

Math.LOG2E (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- `Math.LOG2E`

This constant provides the numeric value for the base-2 logarithm of e, the base of natural logarithms.

By inspection, the resulting value returned is 1.4426950408889634 although this may vary from platform to platform.

Warnings:

- Note that the value of `Math.LOG2E` is approximately the reciprocal of the value of `Math.LN2`. The word approximately is used here, because the mathematical accuracy of JavaScript implementations leaves something to be desired and there are some strange artifacts in some of the calculations.

See also:	Arithmetic constant, Exponent-log function, Floating point constant, `Math` object, `Math.exp()`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.4

ECMA 262 edition 3 –section –15.8.1.4

Math.max() (Function)

The maximum of the two or more input arguments is returned.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0

Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.max(aValue1, aValue2, ...)`
Argument list:	*aValue1*	Some meaningful numeric value
	aValue2	Some meaningful numeric value

Returns the larger of the two or more arguments.

Special boundary conditions that affect the results are:

Argument1	Argument2	Result
Any value	NaN	NaN
Any value	The same value	The value
larger	smaller	Argument1
NaN	Any value	NaN
smaller	larger	Argument2

See also:	`Math` object

Cross-references:

ECMA 262 edition 2 –section –15.8.2.11

ECMA 262 edition 3 –section –15.8.2.11

Math.min() (Function)

The minimum of the two or more input arguments is returned.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.min(aValue1, aValue2, ...)`
Argument list:	*aValue1*	Some meaningful numeric value
	aValue2	Some meaningful numeric value

This function returns the smaller of the two or more arguments.

Special boundary conditions that affect the results are:

Argument1	Argument2	Result
Any value	NaN	NaN
Any value	The same value	The value
larger	smaller	Argument2
NaN	Any value	NaN
smaller	larger	Argument1

See also:	`Math` object

Cross-references:

ECMA 262 edition 2 –section –15.8.2.12

ECMA 262 edition 3 –section –15.8.2.12

Math.PI (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.PI`

This constant provides the numeric value for pi, the ratio of circumference of a circle to its diameter.

The resulting value returned is approximately 3.141592653589793 (to 15 d.p.)

Warnings:

- Note that `Math.sin(Math.PI/2)` may not yield exactly 1.
- Note that `Math.cos(Math.PI)` may not return precisely zero.
- Note that `Math.acos(0)` may not return the same value as `Math.PI`.

See also:	Arithmetic constant, Floating point constant, `Math` object, `Math.acos(),Math.cos(),Math.sin()`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.6

ECMA 262 edition 3 –section –15.8.1.6

Math.pow() (Function)

The result of raising a value to the power of another value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.pow(`*`aValue1`*`, `*`aValue2`*`)`
Argument list:	*`aValue1`*	Some meaningful numeric value
	`aValue2`	Some meaningful numeric value

This function returns the result of raising the first argument to the power of the second.

Special boundary conditions that affect the results are:

Argument1	Argument2	Result
+0	< 0	+infinity
+infinity	< 0	0
+infinity	> 0	+infinity
-0	< 0 and is an even integer	+infinity
-0	< 0 and is an odd integer	-infinity
-infinity	< 0	0
-infinity	> 0 and is an even integer	+infinity
-infinity	> 0 and is an odd integer	-infinity
0	> 0	0
1	Any value	1
< 0 and finite	finite but non integer	NaN
abs(arg) < 1	+infinity	0

Table continued on following page

Argument1	Argument2	Result
abs(arg) < 1	-infinity	+infinity
abs(arg) == 1	+infinity	NaN
abs(arg) == 1	-infinity	NaN
abs(arg) > 1	+infinity	+infinity
abs(arg) > 1	-infinity	0
Any value	0	1
Any value	NaN	NaN
NaN	non zero	NaN

The exact value yielded by this function may vary slightly from implementation to implementation, due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

The example shows a simple binary number converter which exhibits some instability in the most significant bit on some platforms.

Warnings:

- There are many boundary conditions that make this function hard to understand and therefore hard to diagnose if it goes wrong. Check both of the input arguments in case of doubt. They may have evaluated out to a strange boundary condition that yields unexpected results.
- Using MSIE on the Macintosh exhibits a instability when you raise 2 to the power 31 and test the resulting value in a bitwise expression. Netscape works correctly in this circumstance. This is demonstrated in the example.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
document.write(-2 + " - " + binary32(-2) + "<BR>");
document.write(-1 + " - " + binary32(-1) + "<BR>");
document.write(0 + " - " + binary32(0) + "<BR>");
document.write(1 + " - " + binary32(1) + "<BR>");
document.write(2 + " - " + binary32(2) + "<BR>");
document.write(3 + " - " + binary32(3) + "<BR>");
function binary32(aValue)
{
   myArray = new Array(32);

   for(myEnum=0; myEnum<32; myEnum++)
   {
      if(aValue & Math.pow(2, myEnum))
      {
         myArray[31-myEnum] = "1";
      }
      else
      {
```

```
            myArray[31-myEnum] = "0";
         }
      }
      return myArray.join("");
   }
   </SCRIPT>
   </BODY>
   </HTML>
```

See also:	`Math` object, `Math.sqrt()`, Power function, Zero value

Cross-references:

ECMA 262 edition 2 –section –15.8.2.13

ECMA 262 edition 3 –section –15.8.2.13

Math.random() (Function)

Generates a pseudo-random value.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –2.02 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.random()`

The `Math.random()` function generates and returns a pseudo-random value; a positive value between 0 and 1.

The resulting value is chosen randomly (or pseudo randomly) depending on the implementation.

In any case, regardless of how it is selected, it should yield a uniform distribution over the range of possible values.

The exact value yielded by this function may vary slightly from implementation to implementation, due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to generate the random numbers.

Some implementations may provide a way to seed the random number sequence although the ECMAScript standard does not describe this capability.

Some implementations provide predictable series of random numbers that always start at the same seed point.

The algorithm and strategy is implementation dependent and the standard offers no recommendations as to which is best.

Warnings:

- Although this is noted as being available in Netscape 2.02, that only applies to the Unix platform. It wasn't widely available until JavaScript 1.1 was supported in Netscape 3.0 on the remaining platforms.

See also:	`Math` object, Pseudo-random numbers

Cross-references:

ECMA 262 edition 2 –section –15.8.2.14

ECMA 262 edition 3 –section –15.8.2.14

Math.round() (Function)

Rounds to the nearest integer value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape Navigator –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.round(`*`aValue`*`)`
Argument list:	*`aValue`*	A meaningful numeric value

This function returns the value that is closest to the argument and is a mathematical integer. It rounds the input value to the nearest integer value either rounding up or down as necessary.

If the input value is equi-distant from two integer values, the result is rounded up towards positive infinity. If the argument is already an integer, the argument itself is returned.

Special boundary conditions that affect the results are:

Argument	**Result**
+infinity	+infinity
-0.5 < arg < 0.5	0
-infinity	-infinity
0	0
NaN	NaN

Note that `Math.round(3.5)` returns the value 4 while `Math.round(-3.5)` returns the value 3.

See also:	Integer arithmetic, Integer-value-remainder, `Math` object, `Math.ceil()`, `Math.floor()`, `Number()`, Type conversion

Cross-references:

ECMA 262 edition 2 –section –15.8.2.15

ECMA 262 edition 3 –section –15.8.2.15

Math.sec() (Simulated functionality)

The `sec()` function is not available in JavaScript but can be simulated to aid in the porting of existing code.

The example demonstrates how to add the `sec()` method to the `Math` object to provide the capability to calculate secants.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Add the sec function to the Math object
function sec(aValue)
{
   return 1/Math.cos(aValue);
}

// Register the new function
Math.constructor.prototype.sec = sec;

// Test the Math.sec() method
document.write(Math.sec(2));
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Math.cosec(),Math.cot()`

Math.sin() (Function)

The sine of the passed in value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.sin(`*`aValue`*`)`
Argument list:	*`aValue`*	An angle measured in radians

This function returns the sine of the input argument. The argument value must be expressed in radians.

Special boundary conditions that affect the results are:

Argument	Result
+infinity	NaN
-infinity	NaN
0	0
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation, due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

Warnings:

- Note that `Math.sin(Math.PI/2)` may not yield exactly 1.

See also:	`Math` object, `Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.PI`, `Math.tan()`, Trigonometric function

Cross-references:

ECMA 262 edition 2 –section –15.8.2.16

ECMA 262 edition 3 –section –15.8.2.16

Math.sinh() (Simulated functionality)

The `sinh()` function is not available in JavaScript but can be simulated to aid in the porting of existing code.

The example demonstrates how to add the `sinh()` method to the `Math` object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Add the sinh function to the Math object
function sinh(aValue)
{
   var myTerm1 = Math.pow(Math.E, aValue);
   var myTerm2 = Math.pow(Math.E, -aValue);
   return (myTerm1-myTerm2)/2;
}
```

```
// Register the new function
Math.constructor.prototype.sinh = sinh;

// Test the Math.sinh() method
document.write(Math.sinh(2));
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

Math.sqrt() (Function)

The square root of the input argument.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.sqrt(`*`aValue`*`)`
Argument list:	*`aValue`*	A meaningful numeric value

This function computes the square root of the input argument.

Special boundary conditions that affect the results are:

Argument	Result
+infinity	+infinity
0	0
< 0	NaN
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation, due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

See also:	`Math` object, `Math.pow()`

Cross-references:

ECMA 262 edition 2 –section –15.8.2.17

ECMA 262 edition 3 –section –15.8.2.17

Math.SQRT1_2 (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- `Math.SQRT1_2`

This constant returns the numeric value of the square root of 0.5.

The resulting value returned is approximately 0.707106781186548 (to 15.d.p.)

Warnings:

- Note that the value of `Math.SQRT1_2` is approximately the reciprocal of the value of `Math.SQRT2`. The word approximately is used here, because the mathematical accuracy of JavaScript implementations leaves something to be desired and there are some strange artifacts in some of the calculations.

See also:	Arithmetic constant, Floating point constant, Math object, `Math.SQRT2`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.7

ECMA 262 edition 3 –section –15.8.1.7

Math.SQRT2 (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0

Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.SQRT2`

This constant provides the value of the square root of 2.

The resulting value returned is approximately 1.414213562373095 (to 15 d.p.)

Warnings:

- Note that `Math.SQRT1_2` may not be exactly equal to the reciprocal of `Math.SQRT2`.

See also:	Arithmetic constant, Floating point constant, `Math` object, `Math.SQRT1_2`

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.8.1.8

ECMA 262 edition 3 –section –15.8.1.8

Math.tan() (Function)

The tangent of the input argument.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Math.tan(`*`aValue`*`)`
Argument list:	*`aValue`*	An angle measured in radians

This function returns the tangent of the input argument. The argument value must be expressed in radians.

Special boundary conditions that affect the results are:

Argument	Result
+infinity	NaN
-infinity	NaN
0	0
NaN	NaN

The exact value yielded by this function may vary slightly from implementation to implementation due to differences in the underlying precision of the implementations, math routines, and the specific algorithm selected to evaluate this function.

See also:	`Math` object, `Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`, Trigonometric function

Cross-references:

ECMA 262 edition 2 –section –15.8.2.18

ECMA 262 edition 3 –section –15.8.2.18

Mathematics (Definition)

Mathematical support.

The following table summarizes the support for mathematical computation in JavaScript:

Function	Object	Type	Description
`abs()`	Math	Function	Absolute value of a number
`acos()`	Math	Function	Inverse cosine
`asin()`	Math	Function	Inverse sine
`atan()`	Math	Function	Inverse tangent
`atan2()`	Math	Function	Inverse tangent of a slope
`ceil()`	Math	Function	Round up
`cos()`	Math	Function	Cosine of an angle
`E`	Math	Constant	Exponential constant
`exp()`	Math	Function	Exponent function
`floor()`	Math	Function	Round down
`isFinite()`	Global	Function	Test for infinity
`isNaN()`	Global	Function	Test for Not-a-Number
`LN10`	Math	Constant	Natural log of 10
`LN2`	Math	Constant	Natural log of 2

Table continued on following page

Function	Object	Type	Description
`log()`	Math	Function	Natural log of a number
`LOG10E`	Math	Constant	Log to the base 10 of e
`LOG2E`	Math	Constant	Log to the base 2 of e
`max()`	Math	Function	Maximum of two values
`MAX_VALUE`	Number	Constant	Maximum numeric value
`min()`	Math	Function	Minimum of two values
`MIN_VALUE`	Number	Constant	Minimum numeric value
`NaN`	Number	Constant	Not a number
`NEGATIVE_INFINITY`	Number	Constant	Negative infinity
`parseFloat()`	Global	Function	Floating point parser
`parseInt()`	Global	Function	Integer parser
`PI`	Math	Constant	The value of PI
`POSITIVE_INFINITY`	Number	Constant	Positive infinity
`pow()`	Math	Function	A value rased to the power of another
`random()`	Math	Function	A random value
`round()`	Math	Function	Truncating round
`sin()`	Math	Function	Sine of an angle
`sqrt()`	Math	Function	Square root
`SQRT1_2`	Math	Constant	Square root of one half
`SQRT2`	Math	Constant	Square root of two
`tan()`	Math	Function	Tangent of an angle

You should avoid the use of any of these function names as identifiers unless you are intentionally overriding their behavior by adding function properties (methods) to a prototype.

See also:	Arithmetic operator, Error, Integer arithmetic, Integer-value-remainder, Operator, Power function, Range error, Trigonometric function

Matrix() (Filter/visual)

A means of applying sophisticated rotation, translate, and scaling effects to an image using matrix transformation.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `Matrix()`

MAYSCRIPT (HTML Tag Attribute)

An attribute on the `<APPLET>` tag to allow Java to access the JavaScript object space.

This is an HTML tag attribute that can significantly affect the success or failure to run applets and plugins properly.

The `MAYSCRIPT` attribute must be present in the `<APPLET>` tag if you have applets that expect to communicate with JavaScript. This is a way of allowing applets to script under the behest of the web page author who may not be the person who programmed the applet. This way, the author has to know and expect the applet to connect to JavaScript before the applet is able to do so.

Without this attribute, an applet is not able to access the `JSObject` class and communicate with the JavaScript environment.

The `MAYSCRIPT` HTML tag attribute is not reflected into the JavaScript environment.

See also:	JavaScript embedded in Java, `JSObject` object

Measurement units (Definition)

Style position and size properties use measurement units to locate objects on the screen.

Measurement units specify the units of measure of a quantity. These are typically used for length measurement but may also be used for time measurement in aural style sheet properties.

The units include the following:

Unit	Description
`#`	A hash precedes hex color triplet values.
`%`	A percentage of the containing element's value.
`cm`	Absolute measure of a centimeter.
`deg`	A value used for angular positioning of sound sources.
`em`	A floating point value indicating a fractional portion of the length of an em-dash in the current font.
`ex`	A floating point value used to multiply the height of a small x in the current font.
`Hz`	A frequency value for aural style sheets.
`in`	Absolute measure of an inch.
`kHz`	A frequency value for aural style sheets.
`mm`	Absolute measure of a millimeter.
`ms`	A value in milli-seconds (used for aural style durations).
`pc`	Absolute measure of a pica.
`pi`	Absolute measure of a pica (possibly a misprint in some documentation).
`pt`	Absolute measure using a font point size.
`px`	An integer value measured in pixels on the screen.
`s`	A value in seconds (used for aural style durations).

Measurement units referring to a spatial distance on the page are sometimes called Length units.

MediaList object (Object/DOM)

This object is added to DOM level 2 to support media lists in style sheets.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
JavaScript syntax:	N	*myMedialist* = new MediaList()

DOM level 2 specifies the following properties for this object:

- `mediaText`
- `length`

DOM level 2 specifies these methods:

- `item()`
- `deleteMedium()`
- `appendMedium()`

Member (Definition)

Elements within an object.

Members of structures and unions in other languages have some analogy with methods and properties of an object. In a procedural language you can access members of a structure by name. So also can you access properties within an object.

A member name is an identifier.

The notation for accessing structure members in C language and property values in JavaScript is identical:

- `anObject.aProperty`
- `aCStruct.aMember`

There is a variation in JavaScript that allows properties to be called as functions, thus:

- `anObject.aFunctionProperty()`

This notation can also denote methods that belong to an object.

See also:	Method, Namespace

Memory allocation (Definition)

The process of locating and allocating some memory to store a string or object.

See also:	Reference counting, Memory management

Memory leak (Definition)

The consumption of memory that is not recoverable.

When the reference count for an object is zero, the object can be garbage collected. Also, because there are no references to it, you have no handle by which you can reach it, so if it isn't garbage collected, it will waste the space it occupies. When that happens, you have a memory leak.

Memory leaks typically happen in web-based JavaScript when you create and destroy a lot of strings in a loop. Concatenating many strings together and extending one string incrementally is a typical leak-producing technique.

Garbage collection generally only happens in web browsers when the page is refreshed.

See also:	delete, Garbage collection, Memory management, `Object()`, `Option()`, Reference counting, Variable, `Window.setInterval()`, `Window.setTimeout()`

Cross-references:

Wrox *Instant JavaScript* –page –29

Memory management (Definition)

The process of organizing and keeping track of memory allocation and de-allocation.

Memory management in compiled languages tends to be a primary concern of software developers. Because JavaScript is interpreted and is intended for use by designers as well as developers, a great deal of the complexity of memory management is hidden from view.

It is still possible however to design a script that will consume large amounts of memory due to what is called a memory leak.

A memory leak is when you allocate some storage and you don't subsequently relinquish it and make it available to the system again.

A prime example of a memory leak in the context of a JavaScript execution is the allocation of string data to `String` variables. When a new assignment is made, the old storage is unlinked from the variable and some new storage is allocated. This means that the storage management is far simpler because the strings tend to grow longer as new values are assigned. However, in a web browser, the discarded string values continue to sit around in memory until the page is refreshed. At that stage, all page local values are purged and the memory is freed.

You cannot force a garbage collection in a JavaScript execution session in a web browser other than by setting the `location.href` of the current page to itself. This has the side effect of reloading the page from the web server, presenting the user with a possibly ugly transition artifact and increasing net traffic. However, this may be far more preferable than consuming 50 Megabytes of memory every few minutes in the client.

See also:	Garbage collection, Memory leak

MENU object (Object/HTML)

An object that represents the contents of a `<MENU>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myMENU* = *myDocument*.all.*anElementID*
	IE	*myMENU* = *myDocument*.all.tags("MENU")[*anIndex*]
	IE	*myMENU* = *myDocument*.all[*aName*]
	-	*myMENU* = *myDocument*.getElementById(*anElementID*)
	-	*myMENU* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myMENU* = *myDocument*.getElementsByTagName("MENU")[*anIndex*]
HTML syntax:	`<MENU> ... </MENU>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`compact`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

The DOM level 1 specification refers to this as a `MenuElement` object.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`compact`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.0 +	1.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

MENU.compact (Property)

An attribute that controls the display of `<MENU>` items and the amount of space they require on the screen.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myMENU*`.compact`

Refer to:

`DL.compact`

menubar (Property)

An alias for the `window.menubar` property.

Availability:	JavaScript –1.2 Netscape –4.0
Property/method value type:	`Bar object`

JavaScript syntax:	-	menubar
	-	*myWindow*.menubar

See also:	Bar object, Window.menubar

Property attributes:

ReadOnly.

<META> (HTML Tag)

Document meta-information container.

Availability:	JavaScript –1.2 Netscape –4.0	
HTML syntax:	<META HTTP-EQUIV="*aName*" CONTENT="*aValue*">	
Argument list:	*aName*	A pseudo header name
	aValue	A pseudo header value

The <META> tag is a way of adding information about the document to the document. This information is never intended for display to the user but helps the browser and other server side systems (and proxies) to manage the document.

There are many attributes to the <META> tag. We have only covered the ones that are helpful in the context of script development here.

The items that are particularly useful to us are the HTTP_EQUIV and CONTENT attributes. These are used to add information to the HTTP response body. The browser can see these values in the <META> tag and interprets them as if they had been part of the HTTP header.

So:

```
<META HTTP-EQUIV="name" CONTENT="value">
```

is understood to be equivalent to an HTTP header like this:

```
name: value
```

As far as JavaScript is concerned, there are two <META> tags that are useful.

This one defines the default scripting language:

```
<META HTTP-EQUIV="Content-Script-Type" CONTENT="text/JavaScript">
```

If you have access to the server mechanism, you might be able to effect the same things by forcing it to add this header to the response as it goes out:

```
Content-Script-Type: text/JavaScript
```

The second useful <META> tag is used to define the default style definition language. Given all the warnings about JSS not being portable and it having been deprecated, you could use this in your document <HEAD> block:

```
<META HTTP-EQUIV="Content-Style-Type" CONTENT="text/JavaScript">
```

Again, if you have access to the server mechanism, you might be able to add this header to the response as it goes out:

```
Content-Style-Type: text/JavaScript
```

Warnings:

- Since MSIE does not support JSS and Netscape ignores the tag variants, these <META> tags are of limited use.

See also:	<SCRIPT LANGUAGE="...">, <SCRIPT>, <STYLE TYPE="...">, <STYLE>, JavaScript Style Sheets, META object

Cross-references:

Wrox *Instant JavaScript* –page –52

META object (Object/HTML)

An object that represents the contents of a <META> tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	Element object	
JavaScript syntax:	IE	*myMETA* = *myDocument*.all.*anElementID*
	IE	*myMETA* = *myDocument*.all.tags("META")[*anIndex*]
	IE	*myMETA* = *myDocument*.all[*aName*]
	-	*myMETA* = *myDocument*.getElementById(*anElementID*)
	-	*myMETA* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myMETA* = *myDocument*.getElementsByTagName("META")[*anIndex*]
HTML syntax:	<META>	

Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`charset, content, httpEquiv, name, scheme, url`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

These tags and their corresponding object instantiations are used to convey hidden information about the document. This information might be useful to a search engine for example. Sometimes the server uses the `META` information to control the way the pages are cached into a proxy. Likewise, a client browser may use these values to control the local caching and expiry times of a document.

There may be several `META` objects associated with a document. There is no collection object that provides an enumerable set containing only the `META` objects but you can traverse the `document.all[]` collection and extract them in MSIE or use the collection returned by the DOM compliant `document.getElementsByTagName("META")` method which is supported on Netscape 6.0 and recent versions of MSIE. It is possible you might know the unique `ID="..."` HTML tag attribute value in which case you can access the required object directly.

This is not currently supported at all in Netscape 4.0 although, because it is part of the DOM specification, it is added to Netscape 6.0.

Warnings:

- Netscape 6.0 returns an undefined value for the charset property and incorrectly appends it to the content property.
- For this meta tag:
  - `<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">`
- Netscape Navigator 6.0 returns these values:
  - `myMeta.httpEquiv` is defined as "`Content-Type`"
  - `myMeta.content` is defined as "`text/html; charset=8859-1`"
  - `myMeta.charset` is undefined

See also:	`<META>`, `Element` object, `META.charset`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`charset`	-	3.0 +	-	4.0 +	-	-	-	-
`content`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`httpEquiv`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`name`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`scheme`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`url`	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.0 +	1.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

META.charset (Property)

A value containing the character encoding of the content in the `<META>` tag.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myMETA*`.charset`

This would contain the character set being used by the document referred to by the `HREF="..."` HTML tag attribute. For example the value `"iso-8859-1"` is likely to be returned but the local variant of the browser and OS may affect the value you get.

This property might contain a value such as:

```
csISO5427Cyrillic
```

Details of other aliases can be located at the IANA registry.

See also:	`META` object, `META.content`

Web-references:

ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

META.content (Property)

The contents of the VALUE="..." HTML tag attribute belonging to the <META> tag that the object represents.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myMETA*.content

A <META> tag will contain a name-value pair stored in two separate HTML tag attributes. The content property relates to the value part of that name-value pair. The name property is contained in a NAME="..." HTML tag attribute.

The web server hides additional information in header records that the client browser uses, but which are invisible to the user and generally hard to access from JavaScript. The <META> tags support an HTTP-EQUIV="..." HTML tag attribute which, although encoded as part of the document source, will behave as if it were a server response header value. This is also used in conjunction with the content property as an alternative way of forming a name-value pair.

See also:	META.charset, META.httpEquiv

META.httpEquiv (Property)

The contents of the HTTP-EQUIV="..." HTML tag attribute belonging to the <META> tag that the object represents.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myMETA*.httpEquiv

This property will also reflect the value of the NAME-="..." HTML tag attribute if that was used in preference to the HTTP-EQUIV="..." HTML tag attribute. However only one or the other may be present in the <META> tag in the document source.

This property is used in conjunction with the content property to construct a name-value pair.

See also:	META.content

META.name (Property)

The name of the meta information this object describes.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myMETA*.name

This is the value of the `NAME="..."` HTML tag attribute in the `<META>` tag that instantiated this object.

META.scheme (Property)

A describer for the content form.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myMETA*.scheme

The meta data in the tags may use the same names in a `name=value` pair to mean something different according to who defined the meta tag contents.

The scheme property reflects the `SCHEME="..."` HTML tag attribute and defines separate namespaces so the meta information can be interpreted more correctly. It offers a context within which the meta data values are defined.

META.url (Property)

A special MSIE supported property containing the URL associated with a `<META>` tag.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myMETA*.url

Occasionally, most likely in an auto-refresh `META` tag, you will need to specify a URL value. An auto-refresh tag uses a client-pull technique to request a page update automatically after a timed interval.

Metacharacter (Definition)

A special symbolic way of describing some property of a character. Used in regular expressions.

Refer to:

`RegExp` pattern

Method (Definition)

A method is an action that can be performed on an object.

Availability:	ECMAScript edition –2

Functions are implemented in the script interpreter objects and are accessed as methods when they are themselves associated with an object.

Methods are owned by objects. An instance of a class can own some private methods, which it does not share. It can also share methods it inherits from its prototype. Privately owned methods are sometimes called instance methods.

Those functions that are associated with the `Global` object do not need an object prefix to be used. The `Global` object is always present and available and in the scope chain and prototype inheritance tree. Therefore the identifiers for those functions can be resolved easily, and in your script code they appear to be like functions in C language.

When you declare functions in your script, as they are constructed they are associated with the `Global` object and are also available in the same way.

Functions associated with the `Math` object require the `Math` object to be cited when they are called. Because they are visibly associated with the object and are called via the object, they are methods.

You can specifically associate one of your own functions with an object other than the `Global` object. If you do that, then you can refer to the owner object with the variable named `'this'`. It is a special variable that is like the `'self'` variable in Smalltalk.

Because a function is an object and associating it with your own object is by means of a reference to the function object, you can share function code between several objects.

So, a method is simply a function that is written in a particular way that means it works well when associated with an object.

As soon as you start to use the `'this'` variable, then it's likely your function is no longer useful as a stand-alone function and ought to really be called as a method from that point onwards.

Example code:

```
// Function to print an owner object property
function my_name()
{
  document.write(this.name);
}
// Create a new object instance
var myObject = new Object;
// Define the name property for the object
myObject.name = "Example";
// Associate the function so it can be used as a method,
// note that we omit the parentheses
myObject.my_name = my_name;
// Call the function via the method interface
myObject.my_name();
// Call the function normally
my_name();
```

See also: Accessor method, Function, `function( ... ) ...`, `java.util`, Member, Property, Statement, `this`

Cross-references:

ECMA 262 edition 2 –section –4.3.3

ECMA 262 edition 3 –section –4.3.3

Wrox *Instant JavaScript* –page –30

Microsoft TV (TV Set-top Box)

A digital TV set-top box.

This is a more enhanced version of the WebTV box. It may well work on analogue but is really intended for deployment in a digital TV environment.

The triggering and standard HTML support is defined according to the ATVEF platform specifications, which are evolving and are not yet complete but are looking to be the definitive profile that 'Browser in a box' type of system needs to conform to.

Technically, this platform is more advanced than WebTV but it is fundamentally the same.

The JavaScript capabilities in this system would be broadly in line with what you would expect a normal PC browser to cope with but there may be a few limitations here and there. There are also likely to be extensions provided as new object types and possibly some additional methods added to existing objects but these would be quite minimal.

Detecting that you are running in a set-top box may not always be easy. The user agent values may not tell you that you have a DTV environment available and you may need to be a little more clever and test for the existence of specific object classes. You need to do that in a way that does not cause a run-time error of course. It's a good technique to practice to ensure your scripts run reliably regardless of whether they are designed for use with Microsoft TV or not.

See also: ATVEF, WebTV

MIME types (Definition)

An Internet standard way of distinguishing between different kinds of container files.

The MIME types mechanism was originally developed for sending attachments in mail messages. Nowadays it has become a commonplace way of describing the content of a document in a way that many non e-mail client applications can understand.

This is defined as part of the HTML 4.0 standard.

Here is a list of some relevant MIME types for JavaScript programmers:

MIMEType	Description
*	Wildcard match everything
/	Wild card match both parts separately
application/applefile	AppleSingle file
application/AppleLink	AppleLink Package
application/ArcMac	PC ARChive
application/BBEdit	ML Source
application/binary	Application Binary Data
application/Canvas	Canvas Drawing
application/cdf	Channels
application/CodeWarrior	Java Class File
application/Compact_Pro	Compact Pro Archive
application/DeArj	ARJ Archive
application/DiskCopy	Apple DiskCopy Image
application/Envoy	Envoy Document
application/Excel	Lotus Spreadsheet r2.1
application/FileMaker_Pro	FileMaker Pro Database
application/FileMaker_Pro_3	FileMaker Pro Database
application/Finder	OpenType Font
application/FoxBase+	DBase Document
application/fractals	Fractal Image Format
application/futuresplash	FutureSplash Player
application/GraphicConverter	Animated NeoChrome
application/gzip	application/gzip
application/HexEdit	Untyped Binary Data
application/java-archive	Java Archive
application/JPEGView	OS/2 Bitmap
application/mac-binhex40	Binhex File
application/MacAmp	MPEG-1 Layer 3

Table continued on following page

MIMEType	Description
application/MacAnim_Viewer	DL Animation
application/macbinary	MacBinary
application/MacBooz	Zoo Archive
application/MacLHA	LHArc Archive
application/macwriteii	MacWrite Document
application/Microsoft_Word	Word for Windows Template
application/MoviePlayer	DV Video
application/ms-powerpoint	application/MS-PowerPoint
application/msword	Word Document
application/netcdf	Channels
application/octet-stream	Binary Executable
application/oda	ODA Document
application/PageMaker	PageMaker 3 Document
application/pdf	PDF File
application/PF_Encrypt	Private File
application/pgp-keys	PGP Key File
application/Photoshop	PhotoShop Document
application/PictureViewer	OS/2 Bitmap
application/PlayerPro	669 MOD Music
application/postscript	PostScript File
application/pre-encrypted	Pre-encrypted Data
application/QuarkXpress	QuarkXpress Document
application/Replica	Replica Document
application/ResEdit	Resource File
application/rtf	Rich Text Format File
application/sdp	Session Description Protocol
application/self-extracting	Self-Extracting Archive
application/Self_Extracting_Archive	Self-Extracting Archive
application/SimpleText	Apple documentation file
application/smil	SMIL Document
application/SoftWindows	MS-DOS Executable
application/SoundApp	Amiga OctaMed music
application/SoundHack	IRCAM Sound
application/streamingmedia	Standard Streaming Metafile
application/StuffIt	StuffIt Archive
application/StuffIt_Expander	PackIt Archive
application/SunTar	Unix BAR Archive
application/vnd.fdf	Forms Data Format

Table continued on following page

MIMEType	Description
application/vnd.lotus-1-2-3	Lotus 123 Document
application/vnd.lotus-approach	Lotus Approach Document
application/vnd.lotus-freelance	Lotus Freelance Document
application/vnd.lotus-organizer	Lotus Organizer Document
application/vnd.lotus-screencam	Lotus ScreenCam Movie
application/vnd.lotus-wordpro	Lotus WordPro Document
application/vnd.ms-access	Microsoft Access Database
application/vnd.ms-excel	Excel Worksheet
application/vnd.ms-powerpoint	PowerPoint Presentation
application/vnd.ms-schedule	Microsoft Schedule+ Application
application/vnd.rn-realmedia	RealMedia File
application/vnd.rn-realplayer	RealPlayer File
application/vnd.rn-realsystem-rjs	RealSystem Skin
application/vnd.rn-realsystem-rmx	RealSystem Secure Media Clip
application/vnd.rn-rn_music_package	RealJukebox Music Package
application/vnd.rn-rsml	RealSystem ML File
application/waf	Website Archive
application/WordPerfect	WordPerfect PC 4.2 Doc
application/wordperfect5.1	WordPerfect PC 5.1 Doc
application/x-authorware-map	Authorware
application/x-cdf	Channels
application/x-compress	Unix Compressed (.z) Files
application/x-compressed	application/x-compressed
application/x-conference	application/x-conference
application/x-cpio	Unix CPIO Archive
application/x-csh	C Shell Program
application/x-director	Shockwave
application/x-dvi	TeX DVI Document
application/x-excel	application/x-excel
application/x-fortezza-ckl	Compromised Key List
application/x-gocserve	CompuServe Inbound Link to CIM 3.0
application/x-gtar	GNU Tape Archive
application/x-gzip	GZIP File
application/x-hdf	HDF Data File
application/x-JavaScript	A `.js` file containing JavaScript source code
application/x-javascript	JavaScript Program
application/x-javascript-config	JavaScript Config
application/x-javascript-config	JavaScript Config

Table continued on following page

MIMEType	Description
application/x-latex	LaTeX Document
application/x-macbinary	MacBinary File
application/x-netcdf	Channels
application/x-ns-proxy-autoconfig	Proxy Auto-Config
application/x-perl	Perl Program
application/x-pkcs7-crl	Certificate Revocation List
application/x-pkcs7-mime	PKCS7 Encrypted Data
application/x-pkcs7-signature	PKCS7 Signature
application/x-rtsp	Real Time Streaming Protocol
application/x-sdp	Scalable Multicast
application/x-sgml	SGML Document
application/x-sh	Bourne Shell Program
application/x-shar	Unix Shell Archive
application/x-shockwave-flash	Shockwave Flash
application/x-stuffit	Stuffit Archive
application/x-tar	TAR Archive
application/x-tcl	TCL Program
application/x-tex	TeX Document
application/x-texinfo	GNU TeXinfo Document
application/x-x509-ca-cert	Certificates
application/x-zip-compressed	Zip Compressed Data
application/xml	HTML Document
application/zip	ZIP Archives
audio/aiff	AIFF Audio
audio/basic	AU Audio
audio/mid	MIDI
audio/midi	MIDI
audio/mp3	MPEG Movie
audio/mpeg	MPEG audio stream
audio/mpegurl	MP3 PlayLists (.m3u,.pls)
audio/mpg	MP3 Audio
audio/rmf	audio/rmf
audio/scpls	MP3 PlayLists (.m3u,.pls)
audio/vnd.qcelp	QCP Audio
audio/vnd.rn-realaudio	RealAudio Clip
audio/wav	WAV Audio
audio/x-aiff	AIFF Audio
audio/x-midi	MIDI

Table continued on following page

MIMEType	Description
audio/x-mp3	MPEG Movie
audio/x-mpeg	MPEG audio stream
audio/x-mpegurl	MP3 PlayLists (.m3u,.pls)
audio/x-mpg	MP3 Audio
audio/x-pn-realaudio	RealAudio
audio/x-pn-realaudio-plugin	RealPlayer Plugin
audio/x-rmf	audio/x-rmf
audio/x-scpls	MP3 PlayLists (.m3u,.pls)
audio/x-wav	WAV Audio
image/gif	GIF Image
image/ief	IEF image
image/jpeg	JPEG Image
image/pict	PICT Image
image/png	PNG Image
image/tiff	TIFF Image
image/vnd.rn-realflash	RealFlash Clip
image/vnd.rn-realpix	RealPix Clip
image/x-bmp	Windows BMP Image
image/x-cmu-raster	CMU Raster Image
image/x-fits	Flexible Image Transport
image/x-macpaint	MacPaint Image
image/x-macpict	PICT Picture
image/x-MS-bmp	Windows Bitmap
image/x-pbm	Portable Bitmap
image/x-pgm	Portable Graymap
image/x-photo-cd	PhotoCD Image
image/x-photoshop	Photoshop Image
image/x-pict	PICT Image
image/x-png	PNG Image
image/x-portable-anymap	PBM Image
image/x-portable-bitmap	Portable Bitmap
image/x-portable-graymap	Portable Graymap
image/x-portable-pixmap	Portable Pixmap
image/x-ppm	Portable Pixmap
image/x-quicktime	QuickTime Image
image/x-rgb	SGI Image
image/x-sgi	SGI Image
image/x-targa	Targa Truevision Image

Table continued on following page

MIMEType	Description
image/x-tiff	TIFF Image
image/x-xbitmap	X Bitmap Image
image/x-xbm	X-Windows Bitmap
image/x-xpixmap	X-Windows Pixmap
image/x-xpm	X-Windows Pixmap
image/x-xwd	X-Windows Dump
image/x-xwindowdump	X Window Dump Image
image/xbitmap	X Bitmap Image
image/xbm	X Bitmap Image
message/external-body	URL Bookmark
Netscape/Source	Special file type
Netscape/Telnet	Netscape Telnet session
Netscape/tn3270	Netscape TN3270 session
text/cdf	Channels
text/css	Text File
text/html	An HTML document.
text/JavaScript	Text formatted JavaScript source code inside a <SCRIPT> block
text/Jscript	Text formatted JScript source code inside a <SCRIPT> block
text/plain	Form content and other plain text documents
text/url	URL File
text/vbs	Text formatted VBScript source code inside a <SCRIPT> block
text/vbscript	Text formatted VBScript source code inside a <SCRIPT> block
text/vnd.rn-realtext	RealText Clip
text/x-cdf	Channels
text/x-vcard	Visiting Card
text/xml	HTML Document
undefined	UUEncoded Data
video/avi	Microsoft Video
video/flc	FLC Animation
video/mpeg	MPEG video/audio stream
video/msvideo	Microsoft Video
video/quicktime	QuickTime Movie
video/vnd.rn-realvideo	RealVideo Clip
video/x-mpeg	MPEG video/audio stream
video/x-mpeg2	MPEG2 Video
video/x-msvideo	Microsoft Video
video/x-qtc	video/x-qtc
x-world/x-3dmf	QuickDraw 3D File
x-world/x-vrml	VRML File

MIME stands for Multi-part Internet Mail Extension but its usefulness has gone way beyond the scope of a simple extension to the mail protocols.

See also:	`<SCRIPT ARCHIVE="...">`, `<SCRIPT SRC="...">`, `<SCRIPT TYPE="...">`, `<STYLE TYPE="...">`, `Anchor.mimeType`, `Anchor.type`, `blob.blobLink()`, `BUTTON.accept`, `Document.open()`, `Form.encoding`, JavaScript Image Source URL, `LINK.type`, `MimeType` object, `MimeType.type`, `OBJECT.codeType`, `OBJECT.type`, `style.cueAfter`, `style.cueBefore`, text/JavaScript, `XML.type`

Cross-references:

Wrox *Instant JavaScript* –page –42

MimeType object (Object/browser)

An object representing a MIME type.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
JavaScript syntax:	-	*myMimeType* = *myWindow*.navigator.mimeTypes[*anIndex*]
	-	*myMimeType* = navigator.mimeTypes[*anIndex*]
	-	*myMimeType* = *myMimeTypeArray*[*anIndex*]
Argument list:	*anIndex*	A reference to an element in a collection
Object properties:	`description, enabledPlugin, name, suffixes, type`	
Collections:	`suffixes[]`	

The example code fragment will list all the available MIME types supported by the browser.

Example code:

```
// List the available mimeTypes
for(ii=0; ii<navigator.mimeTypes.length; ii++)
{
   document.write(navigator.mimeTypes[ii].suffixes);
   document.write(" - ");
   document.write(navigator.mimeTypes[ii].type);
   document.write(" - ");
   document.write(navigator.mimeTypes[ii].description);
   document.write(" - ");

   if(navigator.mimeTypes[ii].enabledPlugin)
   {
      document.write(navigator.mimeTypes[ii].enabledPlugin.name);
   }
   else
```

```
    {
       document.write("<no_plugin>");
    }
    document.write("<BR>");
}
```

See also:	`MIME` types, `MimeTypeArray` object, `Navigator.mimeTypes[]`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`description`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	ReadOnly
`enabledPlugin`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	ReadOnly
`name`	1.5 +	5.0 +	6.0 +	5.0 +	-	-	ReadOnly
`suffixes`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	-
`type`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	ReadOnly

MimeType.description (Property)

The descriptive text for a MIME type.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myMimeType`*`.description`

This will not be consistent from browser to browser. In some cases there are similarities in the description. In Netscape Navigator there are bugs in the descriptive text content for some MIME types.

Property attributes:

`ReadOnly`.

MimeType.enabledPlugin (Property)

The plugin object that handles this MIME type.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`Plugin object`	
JavaScript syntax:	-	*`myMimeType`*`.enabledPlugin`

You can use this property to check for the existence of an enabled plugin and generate some kind of dialog that gracefully degrades the performance of your web pages for users who need to download and install a vital plugin module.

Property attributes:

`ReadOnly`.

MimeType.name (Property)

The name of this particular MIME type.

Availability:	JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myMimeType`*`.name`

Although the MIME type names are more consistent between browsers than the descriptive text, there are still significant and annoying differences between them. Use with caution as part of your graceful degradation when you find local client support lacks some vital functionality your pages require.

Property attributes:

`ReadOnly`.

MimeType.suffixes[] (Collection)

A list of file type suffixes that contain data of this MIME type.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
JavaScript syntax:	-	*`myMimeType`*`.suffixes`

This is another example of cross browser inconsistencies. In addition, the result of accessing this property is not truly a collection but a comma delimited string which needs to be unpacked into an array. In some cases, the string is empty where there are no appropriate file types.

Any code you develop needs to take account of the separator being a comma in MSIE and a comma+space in Netscape.

Property attributes:

`ReadOnly`.

MimeType.type (Property)

The name of a MIME type.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myMimeType*`.type`

This value is the recognized MIME type value for this object.

See also:	MIME types

Property attributes:

`ReadOnly`.

MimeTypeArray object (Object/browser)

A collection of `MimeType` objects.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
JavaScript syntax:	-	*myMimeTypeArray* `= navigator.mimeTypes`
Object properties:	`length`	

On MSIE, this array contains an element for each supported `MimeType` object that the browser can respond to.

Likewise on Netscape, a collection of somewhat different `MimeType` objects are available.

Warnings:

- MSIE prior to version 5.0 does not support this facility and returns the undefined value even though it has a place holder for the property.
- Note that Netscape 4.7 on Macintosh exhibits some careless `MimeType` instantiation with at least two of the items in this array.

- Item 50 does not exhibit an associative value because the type property for that `MimeType` is undefined. There is no apparent fix because the type property of a `MimeType` object is read-only although there is no error generated when trying to assign a new value to it. The proper type value for this item should be "`application.gzip`".
- Item 64 has a similar problem but just presents a `null` string rather than its index. This `MimeType` represents UU encoded files. It doesn't show up in the list of applications that map to file types in the Netscape preferences.
- This problem may be extant on other platforms, although it is possible that the problems may be related to your user preference settings and the indices that have problems may not be the same number or `MimeType` values.
- The implications are that GZIP and UU encoded files are handled internally by the browser and therefore, although it creates `MimeType` objects, they may not be derived from the preferences settings although changing your preferences may relocate these objects to different positions in the collection.
- Build an enumerator to examine the `MimeType` objects in the collection. You'll see a fragment of script that would do this in the example. There is no real fix for this other than to build some conditional check into any enumeration or iterative loop that examines the MimeTypes array.
- Note that Netscape 6.0 does not enumerate this collection properly and the example does not yield the correct result, in fact it only seems to work with Netscape 4.

Example code:

```
// Inline this script fragment to display all mime types
// supported by your browser (except for NNav 6.0 which exhibits
// a strange bug).

for(myProp in navigator.mimeTypes)
{
   document.write(myProp);
   document.write("<BR>");
}
```

See also:	`Collection` object, `MimeType` object, `Navigator.mimeTypes[]`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`length`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	ReadOnly

MimeTypeArray.length (Property)

The number of discrete MIME types that the browser supports.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`navigator.mimeTypes.length`

Property attributes:

`ReadOnly`.

Refer to:

`Collection.length`

Minima-maxima (Definition)

Limits for values in the environment or script.

The ECMA standard defines some limits for numeric values and the conditions under which it should evaluate numeric values according to the IEEE 754 standard behavior.

The standard provides for the implementation to support some enquiry functions so you can determine these limits.

The IEEE 754 computation logic further provides for error conditions to be handled gracefully by supplying not a number (`NaN`) values, infinity and undefined values in cases where a computation would normally generate a domain error and possibly crash the application.

If JavaScript did not provide such a forgiving environment in which to execute scripts, it is likely that some of the most trivial script errors would result in the hosting environment crashing. In the context of a web browser you would lose the session and have to restart the browser. On a desktop platform, this could be fatal to the OS and you might have to reboot the machine.

See also:	Limits, Range error, Type conversion

Minus (-) (Operator/additive)

Subtract one operand from another.

Availability:	ECMAScript edition –2

See also:	Add (+), Additive operator, Subtract (-)

Cross-references:

ECMA 262 edition 2 –section –11.6.2

ECMA 262 edition 2 –section –11.13

ECMA 262 edition 3 –section –11.6.2

Minus then assign (-=) (Operator/assignment)

Subtract the right value from the left, modifying the left value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0
See also:	Add then assign (+=), Additive operator, Assignment operator, Decrement value (--), LValue, Subtract then assign (-=)

Cross-references:

ECMA 262 edition 2 –section –11.13

ECMA 262 edition 3 –section –11.13

mocha: URL (Request method)

This is a pseudonym for the JavaScript: URL. It is relevant to Netscape and is probably not supported on other browsers.

See also:	URL, javascript: URL

ModElement object (Object/DOM)

A DOM level 1 object that describes a modification to a document.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
JavaScript syntax:	-	`myModElement = new ModeElement()`
Object properties:	`cite, dateTime`	

The MSIE browser implements deletions and insertions as separate object types (`DEL` and `INS` respectively). The DOM level 1 standard includes both in a single object class.

See also:	`DEL` object, `INS` object

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`cite`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`dateTime`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

ModElement.cite (Property)

A URL that references a document that describes why the item was modified.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myModElement`*`.cite`

The URL of the document that describes why the text was marked as modified is noted in this property.

See also:	`DEL.cite`

ModElement.dateTime (Property)

The date and time that the modification occurred.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myModElement`*`.dateTime`

This is the date and time value for when the modification change occurred. If you are maintaining change control down to the sub-document level in a content management system, these values can be defined from change records in the database.

See also:	`DEL.dateTime`

Modulo (Operator/multiplicative)

Modulo operations are called remainder in JavaScript. They may also be called modulus.

See also:	Days in year, Multiplicative expression, Remainder (%)

Money (Definition)

A locale specific value.

See also:	Localization, Currency symbol

Month from time (Time calculation)

A date and time algorithm defined by ECMAScript.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

In an ECMA compliant implementation, given a time measured in milliseconds from 01-January-1970 UTC, the month number can be calculated from the value.

The formula for calculating day number is shown here:

`Day(t) = floor(t/msPerDay)`, where:

t = an instant in time measured in milliseconds relative to 01-January-1970 UTC.

```
msPerDay = 86400000
```

All non-leap years have 365 days with the usual number of days in each month. Leap years have an extra day in February. The calculation shown below uses known leap years and non-leap years to adjust the day numbers and yield the day number of the first day of the given year and then use that to work out the time in milliseconds when the year started:

```
DayFromYear(y) =
365 * (y – 1970) +
floor((y – 1969) / 4) -
floor((y – 1901) / 100) +
floor((y – 1601) / 400)
TimeFromYear(y) = msPerDay * DayFromYear(y)
```

`YearFromTime(t)` = The largest integer `y` to make `TimeFromYear(y)` less than or equal to `t`.

```
DayWithinYear(t) = Day(t) – DayFromYear(YearFromTime(t))
```

The month value is worked out with this formulaic framework:

`MonthFromTime(t)` = lookup according to `DayWithinYear(t)` falling into a range according to the following table:

Greater than	Less than	Month	Name
000	031	0	January
031	059 + InLeapYear(t)	1	February
059 + InLeapYear(t)	090 + InLeapYear(t)	2	March
090 + InLeapYear(t)	120 + InLeapYear(t)	3	April
120 + InLeapYear(t)	151 + InLeapYear(t)	4	May
151 + InLeapYear(t)	181 + InLeapYear(t)	5	June
181 + InLeapYear(t)	212 + InLeapYear(t)	6	July
212 + InLeapYear(t)	243 + InLeapYear(t)	7	August
243 + InLeapYear(t)	273 + InLeapYear(t)	8	September
273 + InLeapYear(t)	304 + InLeapYear(t)	9	October
304 + InLeapYear(t)	334 + InLeapYear(t)	10	November
334 + InLeapYear(t)	365 + InLeapYear(t)	11	December

Note that `MonthFromTime(0)` is 0 which corresponds to Thursday, 01-January-1970.

Example code:

```
// Work out a month number from a time value
var msPerDay = 86400000;
var myMilliseconds = Number(new Date());
document.write(monthFromDayNumber(myMilliseconds));
// Month from day number
function monthFromDayNumber(aMillisecondTime)
{
   var myMonthLookup = new Array();

   var myDayWithinYear = dayWithinYear(aMillisecondTime);
   var myYearFromTime  = yearFromTime(aMillisecondTime);
   var myLeapNumber    = +inLeapYear(myYearFromTime);
   myMonthLookup[0]  =   0;
   myMonthLookup[1]  =  31;
   myMonthLookup[2]  =  59 + myLeapNumber;
   myMonthLookup[3]  =  90 + myLeapNumber;
   myMonthLookup[4]  = 120 + myLeapNumber;
   myMonthLookup[5]  = 151 + myLeapNumber;
   myMonthLookup[6]  = 181 + myLeapNumber;
   myMonthLookup[7]  = 212 + myLeapNumber;
   myMonthLookup[8]  = 243 + myLeapNumber;
   myMonthLookup[9]  = 273 + myLeapNumber;
   myMonthLookup[10] = 304 + myLeapNumber;
   myMonthLookup[11] = 334 + myLeapNumber;

   for(var ii=0; ii<12; ii++)
   {
      if(myDayWithinYear < myMonthLookup[ii])
      {
```

```
            return ii;
         }
      }
      return 12;
   }
   // Work out day number within year based on time value
   function dayWithinYear(aMilliseconds)
   {
      var myDayNumber     = dayNumber(aMilliseconds);
      var myYearFromTime  = yearFromTime(aMilliseconds);
      var myDayFromYear   = dayFromYear(myYearFromTime);
      var myDayWithinYear = myDayNumber - myDayFromYear;
      return myDayWithinYear;
   }
   // Return year number based on time value
   function yearFromTime(aMilliseconds)
   {
      var myStartYear = 1970;
      while(timeFromYear(myStartYear) < myMilliseconds)
      {
         myStartYear++
      }
      return myStartYear-1;
   }
   // Work out milliseconds at start of year
   function timeFromYear(aYear)
   {
      var myTime = msPerDay * dayFromYear(aYear);
      return myTime;
   }
   // Work out day number from milliseconds
   function dayNumber(aMillisecondTime)
   {
      var myDay = Math.floor(aMillisecondTime/msPerDay);
      return myDay;
   }
   // Day from year function
   function dayFromYear(aYear)
   {
      var myDay = 365 * (aYear - 1970)             +
      Math.floor((aYear - 1969) / 4)   -
      Math.floor((aYear - 1901) / 100) +
      Math.floor((aYear - 1601) / 400);
      return myDay;
   }
   // Flag a leap year with a Boolean value
   function inLeapYear(aYear)
   {
      if((aYear % 4) != 0)
      {
         return false;
      }
      if(((aYear % 100) != 0) ||
   ((aYear % 400) == 0))
      {
         return true;
      }
      return false;
   }
```

See also:	Date from time, Date number, Day from year, Day number, Day within year, Month number, Year from time

Cross-references:

ECMA 262 edition 2 –section –15.9.1.4

ECMA 262 edition 3 –section –15.9.1.4

Month number (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

In ECMA compliant implementations, months are identified by an integer in the range 0 to 11, inclusive.

The month number is calculated by taking the number of the day at the target time and then taking the day number at the start of that year. The difference is the day number within the year, which can then be used via a compare and lookup mechanism to deduce the month. The `InLeapYear()` method comes into play to offset the day number by one when it is a leap year.

See also:	Day number, Day within year, Month from time, Time range

Cross-references:

ECMA 262 edition 2 –section –15.9.1.4

ECMA 262 edition 3 –section –15.9.1.4

MotionBlur() (Filter/visual)

An enhanced motion blur artefact that replaces the older `Blur()` filter functionality.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

filter – `MotionBlur()`

Mouse events (Definition)

Mouse events are part of the browser's event handling complex and are triggered by physical interaction with the mouse.

These events correspond to user actions as the mouse is moved over various elements on the screen or as the user clicks the mouse buttons.

Note that some of these events are triggered in multiples as a result of a single action.

For example, as a mouse button is pressed, a `mouse-down` event is fired. As it is released, a `mouse-up` and a `click` event are fired. A `click` event is only fired once for each `mouse-down` and `mouse-up` pair.

A `double-click` requires two `down-up` cycles within quick succession but the `mouse-down` and `mouse-up` events will also fire.

The DOM standard refines the event model and creates a specific object class to handle Mouse Events. The event model will evolve as more browsers take on the DOM specified model.

See also:	`onClick`, `onDblClick`, `onMouseDown`, `onMouseDrag`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`

MouseEvent object (Object/DOM)

This is part of the DOM level 2 mouse event set.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
JavaScript syntax:	N	*myMouseEvent* = new MouseEvent()
Object properties:	`altKey, bubbles, button, cancelable, clientX, clientY, cllientX, ctrlKey, currentTarget, detail, eventPhase, metaKey, relatedTarget, screenX, screenY, shiftKey, target, timeStamp, type, view`	
Object methods:	`initEvent(), initMouseEvent(), initUIEvent(), preventDefault(), stopPropagation()`	

The availability of the `MouseEvent` object handling can be determined with the `Implementation.hasFeature()` method call.

The available set of events is defined by HTML 4.0 and DOM level 0 with some additional events having been added. These event types are enumerated in the DOM level 2 specification and are:

- `click`
- `mousedown`
- `mouseup`
- `mouseover`
- `mousemove`
- `mouseout`

The contextual information is carried in the detail property which is inherited from the `UIEvent` object. This value is incremented for each complete mouse click cycle but is reset to zero if the mouse is moved, even if that happens between `mousedown` and `mouseup`.

This kind of event is a scenario where event bubbling is likely to be useful because the mouse may click on a pixel sized item which is a child of the object that is really receiving a higher level event.

See also:	`AbstractView object`, `Event` object, `Implementation.hasFeature()`, `onClick`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `UIEvent` object

Property	JavaScript	JScript	Nav	IE	Opera	DOM	Notes
`altKey`	1.5 +	-	6.0 +	-	-	2 +	Warning, ReadOnly
`bubbles`	1.5 +	-	6.0 +	-	-	2 +	-
`button`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`cancelable`	1.5 +	-	6.0 +	-	-	2 +	-
`clientX`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`clientY`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`cllientX`	1.5 +	-	6.0 +	-	-	2 +	-
`ctrlKey`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`currentTarget`	1.5 +	-	6.0 +	-	-	2 +	-
`detail`	1.5 +	-	6.0 +	-	-	2 +	-
`eventPhase`	1.5 +	-	6.0 +	-	-	2 +	-
`metaKey`	1.5 +	-	6.0 +	-	-	2 +	Warning, ReadOnly
`relatedTarget`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`screenX`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`screenY`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`shiftKey`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`target`	1.5 +	-	6.0 +	-	-	2 +	-
`timeStamp`	1.5 +	-	6.0 +	-	-	2 +	-
`type`	1.5 +	-	6.0 +	-	-	2 +	-
`view`	1.5 +	-	6.0 +	-	-	2 +	-

Method	JavaScript	JScript	Nav	IE	Opera	DOM	Notes
`initEvent()`	1.5 +	-	6.0 +	-	-	2 +	-
`initMouseEvent()`	1.5 +	-	6.0 +	-	-	2 +	-
`initUIEvent()`	1.5 +	-	6.0 +	-	-	2 +	-
`preventDefault()`	1.5 +	-	6.0 +	-	-	2 +	-
`stopPropagation()`	1.5 +	-	6.0 +	-	-	2 +	-

MouseEvent.altKey (Property)

A Boolean value that represents the state of the `[alt]` key.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*`myMouseEvent`*`.altKey`

When an event is being processed, you may want to know the state of the `[alt]` key on the keyboard.

This Boolean property returns `true` when the `[`*`alt`*`]` key is pressed and `false` when it is not.

This property reflects the state of the `[`*`alt`*`]` key at the instant when the event was triggered. The user may have released the `[`*`alt`*`]` key in the meantime so you should not assume that if the `[`*`alt`*`]` key was pressed earlier on that it is still pressed when the event handler is being executed.

Warnings:

- The key may not always be labelled "ALT" on the keyboard.

See also:	`Event.altKey`

Property attributes:

`ReadOnly`.

MouseEvent.button (Property)

The mouse button that was pressed to trigger the event.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myMouseEvent`*`.button`

If you need to determine which mouse button was pressed to trigger the event, this property will contain a value to indicate the button.

Note that on a Macintosh, this value is meaningless as there is only one button on an Apple Macintosh mouse.

Third party mice can be added to Apple Macintosh systems to provide multiple (2 or 3) button functionality but they are by no means commonplace and it is probably safe to assume only a 1 button mouse on the Macintosh platform.

The values for the buttons are defined to cope with 1, 2 or 3 button mice.

The following values correspond with mouse buttons:

- 0 –No mouse button was pressed when the event was triggered
- 1 –The left button was pressed
- 2 –The right button was pressed
- 3 –The middle button was pressed

See also:	`Event.button`, `Event.which`

Property attributes:

`ReadOnly`.

MouseEvent.clientX (Property)

Mouse position relative to the web page.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myMouseEvent`*`.clientX`

This is the horizontal position of the mouse when the event was triggered. The position is calculated relative to the visible document area within the window or frame the mouse was positioned in when the event triggered.

See also:	`Event.clientX`

Property attributes:

`ReadOnly`.

MouseEvent.clientY (Property)

Mouse position relative to the web page.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myMouseEvent*`.clientY`

This is the vertical position of the mouse when the event was triggered. The position is calculated relative to the visible document area within the window or frame the mouse was positioned in when the event triggered.

See also:	`Event.clientY`

Property attributes:

`ReadOnly`.

MouseEvent.ctrlKey (Property)

A Boolean value that represents the state of the `[control]` key.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myMouseEvent*`.ctrlKey`

When an event is being processed, you may want to know the state of the `[`*control*`]` key on the keyboard.

This Boolean property returns `true` when the `[`*control*`]` key is pressed and `false` when it is not.

This property reflects the state of the `[`*control*`]` key at the instant when the event was triggered. The user may have released the key in the meantime so you should not assume that if the `[`*control*`]` key was pressed earlier on that it is still pressed when the event handler is being executed.

See also:	`Event.ctrlKey`

Property attributes:

`ReadOnly`.

MouseEvent.initMouseEvent() (Method)

After creating a `MouseEvent` object, it must be initialised with this method call.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
JavaScript syntax:	N	*myMouseEvent*.initMouseEvent(*aType*, *aBubble*, *aCancel*, *aView*, *aDetail*, *aScrnX*, *aScrnY*, *aClntX*, *aClntY*, *aCtrl*, *anAlt*, *aShift*, *aMeta*, *aButton*, *aRelTarg*)
Argument list:	*aType*	A string value describing the event type
	aBubble	A boolean flag indicating whether the event can bubble
	aCancel	A boolean flag indicating whether the event can be cancelled
	aView	A reference to an `AbstractView` object
	aDetail	A value describing the event detail
	aScrnX	A screen X coordinate value
	aScrnY	A screen Y coordinate value
	aClntX	A client X coordinate value
	aClntY	A client Y coordinate value
	aCtrl	A boolean value indicating the state of the control key
	anAlt	A boolean value indicating the state of the alt key
	aShift	A boolean value indicating the state of the shift key
	aMeta	A boolean value indicating the state of the meta key
	aButton	A numeric value indicating which button is being emulated
	aRelTarg	A reference to an `EventTarget` object

A new event object is manufactured by calling the `DocumentEvent.createEvent()` method. That event should have been defined with a type specified as "`MouseEvent`". If it was, then it will support an `initMouseEvent()` method. This must be called before the event is dispatched otherwise the event object will not contain enough information for the event dispatcher/handler to make sense of it and route it to the correct target objects.

Two boolean argument values define whether the event will be allowed to be cancelled and what type of propagation to use (`bubble` or `capture`).

The view argument refers to an `AbstractView` object which DOM level 2 describes and which may not yet be well supported by any browser.

The detail value can be used to pass context information into the event handling chain.

The screen and client coordinate values are measured in pixels and must be specified as numeric values.

Five boolean values then define the state of keyboard keys when the event was assumed to be triggered.

Finally a reference to a related target object is passed.

See also:	`AbstractView object`, `EventTarget` object, `UIEvent.initUIEvent()`

MouseEvent.metaKey (Property)

A Boolean value that represents the state of the [*meta*] key.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myMouseEvent*.metaKey

When an event is being processed, you may want to know the state of the [*meta*] key on the keyboard.

This Boolean property returns `true` when the [*meta*] key is pressed and `false` when it is not.

This property reflects the state of the [*meta*] key at the instant when the event was triggered. The user may have released the [*meta*] key in the meantime so you should not assume that if the [*meta*] key was pressed earlier on that it is still pressed when the event handler is being executed.

Warnings:

- The key may not always be labelled "META" on the keyboard.

Property attributes:

`ReadOnly`.

MouseEvent.relatedTarget (Property)

A related `EventTarget` object is referred to.

Availability:	DOM level –2 JavaScript –1.5 Netscape Navigator version –6.0	
Property/method value type:	`EventTarget object`	
JavaScript syntax:	N	`myMouseEvent.relatedTarget`

The particular `EventTarget` referred to depends on the event type and context. For example, the standard suggests this might be the object the mouse has just moved off when a `mouseOver` is being processed.

The `target` and `relatedTarget` property values would be exchanged vice versa if a `mouseOut` event were being processed.

See also:	`EventTarget` object

Property attributes:

`ReadOnly`.

MouseEvent.screenX (Property)

Mouse position relative to the screen display.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myMouseEvent*.screenX

You may need to know the position of the mouse relative to the screen display coordinates and not the browser window or objects within it. This property provides the horizontal coordinate of the mouse within the screen.

See also:	`Event.screenX`

Property attributes:

`ReadOnly`.

MouseEvent.screenY (Property)

Mouse position relative to the screen display.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myMouseEvent*.screenY

You may need to know the position of the mouse relative to the screen display coordinates and not the browser window or objects within it. This property provides the vertical coordinate of the mouse within the screen.

See also:	`Event.screenY`

Property attributes:

`ReadOnly`.

MouseEvent.shiftKey (Property)

A Boolean value that represents the state of the `[shift]` key.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myMouseEvent*`.shiftKey`

When an event is being processed, you may want to know the state of the `[`*shift*`]` key on the keyboard.

This Boolean property returns `true` when the `[`*shift*`]` key is pressed and `false` when it is not.

This property reflects the state of the `[`*shift*`]` key at the instant when the event was triggered. The user may have released the key in the meantime so you should not assume that if the `[`*shift*`]` key was pressed earlier on that it is still pressed when the event handler is being executed.

See also:	`Event.shiftKey`

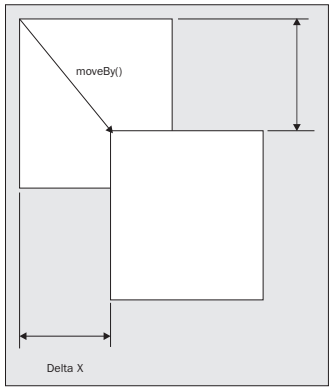
Property attributes:

`ReadOnly`.

moveBy() (Method)

An alias for the `window.moveBy()` method.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`moveBy(`*anOffsetX*`, `*anOffsetY*`)`
	-	*myWindow*`.moveBy(`*anOffsetX*`, `*anOffsetY*`)`
Argument list:	*anOffsetX*	A distance in pixels
	anOffsetY	A distance in pixels

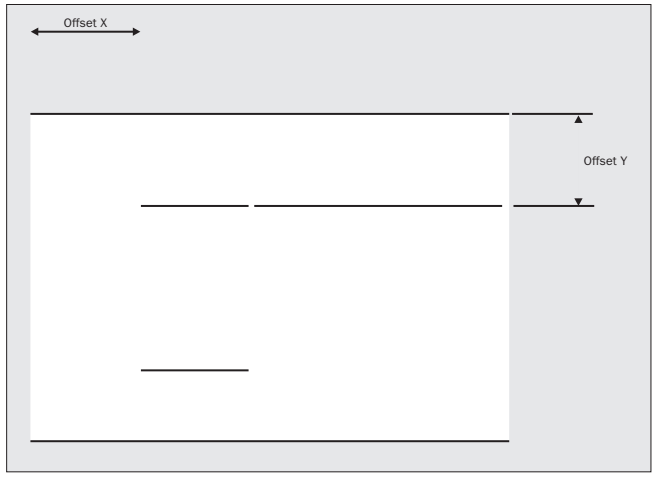


See also:	`Window.onmove`,`Window.moveBy()`

moveTo() (Method)

An alias for the `window.moveTo()` method.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`moveTo(`*`aCoordX`*`, `*`aCoordY`*`)`
	-	*`myWindow`*`.moveTo(`*`aCoordX`*`, `*`aCoordY`*`)`
Argument list:	*`aCoordX`*	A position in pixels
	`aCoordY`	A position in pixels



See also:	`Window.onmove`,`Window.moveTo()`

MSIE (Web browser)

An acronym for the Microsoft web browser.

See also:	JScript version, Undocumented features, Web browser, Internet Explorer

Multi-byte character (Definition)

Character sets using more than 8 bits to represent a code point.

JavaScript natively supports the Unicode character set. This means it is multi-byte character aware at the most fundamental level.

Unicode and localization issues are related to one other and the one depends on the other. You cannot effectively localize an environment properly without multi-byte character sets. The extensions to ASCII to make it an international character set were workable but somewhat inconvenient. All manner of escape sequences were required and ultimately, it consumes more space than a multi-byte character would have.

Unicode extended the width of the code points from 8 bits to 16. This allows the character set to increase from 256 to 65536 code points. To all intents and purposes, this is sufficient to encode every glyph and character share required by all languages world-wide.

However, there may still be some limitations in the encoding used for Far-Eastern language variants. These may be addressed in the Unicode version 3.0 standard.

See also:	Character set, Localization, Unicode

Multi-dimensional arrays (Definition)

Useful techniques for manipulating matrices for math problems.

Multi-dimensional arrays are not supported directly in JavaScript but you can construct them with arrays of arrays.

An array can refer to another array with one of its elements. This means you can build multi-dimensional arrays, which are useful for working out 3D transformations. You probably wouldn't implement a renderer or 3D modeller in JavaScript. However, you might have a Java applet that takes rotation values or perhaps a 3D viewing plugin of some sort that does the hard work.

Warnings:

- The arrays are not truly multi-dimensional and you must be careful to construct them properly and avoid damaging them inadvertently.

Example code:

```
// An example shamelessly stolen from Wrox Instant JavaSscript
// Create a 2x2 array and store an identity matrix in it.
var matrix = new Array(2);
matrix[0] = new Array(2);
matrix[1] = new Array(2);
matrix[0][0] = 1;
matrix[1][0] = 0;
matrix[0][1] = 0;
matrix[1][1] = 1;
```

See also:	Array index delimiter (`[]`), Copying objects

Cross-references:

Wrox *Instant JavaScript* –page –16

Multi-line comment (Definition)

Comment blocks that span several lines of script source text.

The character sequence `/*` introduces a multi-line comment. That comment block is terminated by the next occurrence of the `*/` character sequence. This means you cannot nest these comments within one another. That is arguably bad coding technique anyway.

If you are used to C language you will most likely have used conditional compilation blocks controlled via the pre-processor. Most implementations of JavaScript do not support this, however one or two recent implementations, especially those that operate outside the context of a web browser, are beginning to introduce many C language-like facilities. One of those is the pre-processor with its macro directives. This is particularly useful to developers even though it is somewhat non-standard.

Multiple line comments are replaced with a single line terminator during the interpretation phase. When the `/* ... */` does not have a line terminator inside, it is simply removed prior to interpretation of the remaining line content.

Everything between the start and end of a multi-line comment is ignored.

Warnings:

- You cannot nest these kinds of comment delimiters within one another.

See also:	Comment, Comment (`//` and `/* ... */`), Lexical convention, Line

Cross-references:

ECMA 262 edition 2 –section –7.3

ECMA 262 edition 3 –section –7.4

Wrox *Instant JavaScript* –page –17

Multiplicative expression (Definition)

An expression containing a multiply or divide operator.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

Multiplicative expressions use the multiplicative operators to yield a result by operating on two other values, which may themselves be nested.

See also:	Divide (/), Expression, Modulo, Multiplicative operator, Remainder (%)

Cross-references:

ECMA 262 edition 2 –section –11.5

ECMA 262 edition 3 –section –11.5

Multiplicative operator (Definition)

A multiply or divide operator.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

Multiplicative operators are those that require multiplication or division to evaluate their result.

The following table lists all operators that are multiplicative and those, which are classified under other categories, which are also multiplicative, by implication:

Operator	Description
`%`	Remainder
`*`	Multiply
`/`	Divide
`*=`	Multiply and assign to an LValue
`/=`	Divide and assign to an LValue
`%=`	Remainder and assign to an LValue

Operands that are used with multiplicative operators must have numeric values or be convertible using the `valueOf()` method.

Warnings:

- The order of evaluation of the operands is described in the ECMA standard and a fully compliant implementation should evaluate from left to right. This means that the expression:
- *myFunctionA()* * *myFunctionB()*
- should evaluate *myFunctionA()* before *myFunctionB()* and any side effects of executing *myFunctionA()* will precede those for *myFunctionB()*.
- However, you should test this and not rely on it being portable across implementations.
- Beware of expressions like this:
  - `0 * myFunction()`
- Even though you may not have put the zero in as a constant, the operand on the left may evaluate to be zero. Some implementations may provide performance enhancements that depend on eliminating unnecessary computation. A zero multiplied by any other value will yield a zero. The function may never be executed.
- If you really want *myFunctionA()*, *myFunctionB()* or *myFunction()* to be executed reliably and in a portable manner, you should evaluate them outside of the expression and assign their results to variables, which you can then use in the expression in their place.
- There is a very marginal performance hit but the scripts will execute more reliably across a wider variety of platforms.

See also:	Arithmetic operator, Associativity, Binary operator, Divide (/), Divide then assign (/=), Expression, Multiplicative expression, Multiply (*), Multiply then assign (*=), Operator, Operator Precedence, Remainder (%), Remainder then assign (%=)

Cross-references:

ECMA 262 edition 2 –section –11.5

ECMA 262 edition 3 –section –11.5

Multiply (*) (Operator/multiplicative)

Multiply one operand by another.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*anOperand1 * anOperand2*
Argument list:	*anOperand1*	A value to be multiplied
	anOperand2	The multiplier value

The * operator performs multiplication, producing the product of its operands.

The multiplication is commutative. That means the operands being multiplied together can be arranged in any order without affecting the outcome as long as they are at the same precedence level. However, the multiplication may not always be associative in ECMAScript compliant interpreters due to finite precision in the evaluations. This means that placing parentheses around the operands may affect the outcome.

For example `(A * B) * C` may not evaluate identically to `A * (B * C)` to associative artifacts but `A * B * C` should be identical to `B * C * A` because of commutation.

The rules of floating point multiplication should be governed by the rules of IEEE 754 double-precision arithmetic.

If either operand is `NaN` then the result will be `NaN`.

The sign of the result is positive if both operands have the same sign, negative if the operands have different signs.

Multiplication of an infinite value by zero results in `NaN`.

Multiplication of infinity by infinity results in infinity. The sign is determined by the sign rules as normal.

Multiplying infinity by a finite non-zero value results in a signed infinity. The sign is determined as normal.

Otherwise, where neither an infinite value or `NaN` is involved, the product is computed and rounded to the nearest representable value. If the magnitude of the result is larger than the largest value the interpreter can cope with, an infinity of the appropriate sign is substituted. If the magnitude is too small to be represented, then a zero value is substituted.

Note that internally, zero values can be negative or positive and the sign may affect the result of subsequent computations.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

See also:	Associativity, Multiplicative operator, Multiply then assign (*=), Operator Precedence

Cross-references:

ECMA 262 edition 2 –section –11.5.1

ECMA 262 edition 2 –section –11.13

ECMA 262 edition 3 –section –11.5.1

Multiply then assign (*=) (Operator/assignment)

Multiply two operands storing the result in the first.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`anOperand1 *= anOperand2`*
Argument list:	*`anOperand1`*	A value to be multiplied and then assigned into
	`anOperand2`	A multiplier value

Multiply the left operand by the right operand and assign the result to the left operand.

This is functionally equivalent to the expression:

`anOperand1` = *`anOperand1`* * *`anOperand2`*;

Although this is classified as an assignment operator it is really a compound of an assignment and a multiplicative operator.

The associativity is right to left.

Refer to the operator precedence topic for details of execution order.

The new value of *`anOperand1`* is returned as a result of the expression.

Warnings:

- The operand to the left of the operator must be an `LValue`. That is, it should be able to take an assignment and store the value.

See also:	Assign value (=), Assignment expression, Assignment operator, Associativity, LValue, Multiplicative operator, Multiply (*), Operator Precedence

Cross-references:

ECMA 262 edition 2 –section –11.13

ECMA 262 edition 3 –section –11.13

MutationEvent object (Object/DOM)

A notification that the document content has changed should trigger a mutation event which is described in one of these objects.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
JavaScript syntax:	N	*myMutationEvent* = new MutationEvent()
Object properties:	attrChange, attrName, bubbles, cancelable, currentTarget, eventPhase, newValue, prevValue, relatedNode, target, timeStamp, type	
Class constants:	ADDITION, MODIFICATION, REMOVAL	
Object methods:	initEvent(), initMutationEvent(), preventDefault(), stopPropagation()	

The availability of the `MutationEvent` object handling can be determined with the `Implementation.hasFeature()` method call.

These event types are enumerated in the DOM level 2 specification and are:

- `DOMSubtreeModified`
- `DOMNodeInserted`
- `DOMNodeRemoved`
- `DOMNodeRemovedFromDocument`
- `DOMNodeInsertedIntoDocument`
- `DOMAttrModified`
- `DOMCharacterDataModified`

The DOM level 2 event module specification doesn't describe the binding of these events to event handlers so although this event model is implemented in Netscape 6.0, you may need to explore the event naming conventions to make effective use of it. Taking the event names and placing the 'on' prefix in front of them and using that as a property name to which you can attach a handler function may work. The DOM level 2 event module also provides an `EventListener` registration which allows you to register event types with `EventTarget` objects, using the `addEventListener()` method.

The contextual information is carried in the `detail` property, and when it is present, will usually describe a reference to a node object or an attribute value.

When the document content is modified, a `MutationEvent` object is instantiated to carry a description of that change to the event handler.

Mutation events cannot be cancelled. This is because the DOM interface would become unwieldy if the document changes were not properly completed. This may change later when the DOM standard introduces transaction handling, although the DOM level 2 event specification does not go to great lengths to explain in detail how that is likely to be implemented.

A cascading effect is very likely with a single DOM tree change causing a number of subsequent mutation events to be fired as lower portions of the tree are affected. The standard does not mandate any particular ordering of these events and leaves it to the implementation to control the sequence. This suggests that the cascaded mutations may occur in a different sequence depending on the browser. You should therefore design your event handler so that it can be called in a re-entrant and random orders and that there should be no dependency on things being traversed in a predictable sequence.

See also:	DOM, DOM –Level 2, DOM Events, Event management, Event model, Event-driven model, `EventListener` object, `EventTarget.addEventListener()`, `Implementation.hasFeature()`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`attrChange`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`attrName`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`bubbles`	1.5 +	-	6.0 +	-	-	2 +	-
`cancelable`	1.5 +	-	6.0 +	-	-	2 +	-
`currentTarget`	1.5 +	-	6.0 +	-	-	2 +	-
`eventPhase`	1.5 +	-	6.0 +	-	-	2 +	-
`newValue`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`prevValue`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`relatedNode`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`target`	1.5 +	-	6.0 +	-	-	2 +	-
`timeStamp`	1.5 +	-	6.0 +	-	-	2 +	-
`type`	1.5 +	-	6.0 +	-	-	2 +	-

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`initEvent()`	1.5 +	-	6.0 +	-	-	2 +	-
`initMutationEvent()`	1.5 +	-	6.0 +	-	-	2 +	-
`preventDefault()`	1.5 +	-	6.0 +	-	-	2 +	-
`stopPropagation()`	1.5 +	-	6.0 +	-	-	2 +	-

MutationEvent.attrChange (Property)

The value in this property describes the kind of change that has taken place when the mutation event was an attribute change.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myMutationEvent`*`.attrChange`

This value is meaningful for `DOMAttrModified` events.

The `MutationEvent` class provides the following constant values for testing attribute change property values:

Value	Symbolic name
1	MODIFICATION
2	ADDITION
3	REMOVAL

See also:	`Attribute` object, `MutationEvent.newValue`, `MutationEvent.prevValue`, `Node.attributes[]`

Property attributes:

`ReadOnly`.

MutationEvent.attrName (Property)

The name of a node attribute that has changed for a `DOMAttrChange` event.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myMutationEvent*`.attrName`

This is a string value containing the name of an attribute that has been changed. The old and new values are available in the `prevValue` and `newValue` properties of the `MutationEvent` if you need to inspect them.

See also:	`Attribute` object, `MutationEvent.newValue`, `MutationEvent.prevValue`, `Node.attributes[]`

Property attributes:

`ReadOnly`.

MutationEvent.initMutationEvent() (Method)

After creating a `MutationEvent` object, it must be initialized with this method call.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
JavaScript syntax:	N	*myMutationEvent*.initMutationEvent(*aType*, *aBubble*, *aCancel*, *aNode*, *aPrev*, *aNew*, *aName*)
Argument list:	*aType*	A string containing the event type
	aBubble	A boolean value indicating whether the event can bubble
	aCancel	A boolean value indicating whether the event can be cancelled
	aNode	A reference to a related `Node` object
	aPrev	A string containing the previous value
	aNew	A string containing the new value
	aName	A string containing the name of an attribute

A new event object is manufactured by calling the `DocumentEvent.createEvent()` method. That event should have been defined with a type specified as "`MutationEvent`". If it was, then it will support an `initMutationEvent()` method. This must be called before the event is dispatched otherwise the event object will not contain enough information for the event dispatcher/handler to make sense of it and route it to the correct target objects.

Two boolean argument values define whether the event will be allowed to be cancelled and what type of propagation to use (`bubble` or `capture`).

You can add a reference to a related node and can also define previous and new values if you are simulating an attribute change. Finally for an attribute change, the attribute name can be specified.

See also:	`Event.target`,`EventTarget object`,`Node object`

MutationEvent.newValue (Property)

When an attribute changes, the new attribute value is available here.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myMutationEvent*.newValue

You can use this value to check that the new attribute value is appropriate and if necessary modify it to ensure it falls within your require range. You can compare it with the previous value if need be.

See also:	`MutationEvent.attrChange`, `MutationEvent.attrName`, `MutationEvent.prevValue`

Property attributes:

`ReadOnly`.

MutationEvent.prevValue (Property)

When an attribute changes, the old attribute value is available here.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myMutationEvent*`.prevValue`

This value preserves the old attribute value so you can restore it into the attribute if you inspect the new value and find that it is inappropriate.

See also:	`MutationEvent.attrChange`, `MutationEvent.attrName`, `MutationEvent.newValue`

Property attributes:

`ReadOnly`.

MutationEvent.relatedNode (Property)

A `Node` object is referred to here which can be used to perform contextual examination of the event.

Availability:	DOM level –2 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Node object`	
JavaScript syntax:	N	*myMutationEvent*`.relatedNode`

For attribute changes, this should contain a reference to the node that owned the attribute that was changed. Other operations such as node addition or removal may contain parent nodes or indeed the node that has just been added during an insertion.

See also:	`Node` object, `Node.parentNode`

Property attributes:

`ReadOnly`.

name (Property)

An alias for the `window.name` property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`name`
	-	*`myWindow.name`*
HTML syntax:	`<FRAME NAME="...">Window.open(...)`	
Argument list:	*`aName`*	A name for the window
	`aString`	A string value containing the new name for the window
	`aURL`	A URL to load into the window

Refer to:

`Window.name`

NAME="..." (HTML Tag Attribute)

An HTML tag attribute that names an object.

Many objects are identified in the DOM hierarchy of the web browser by means of their `name` property. This value is defined as an HTML tag attribute.

Some objects can be accessed using an `ID="..."` HTML tag attribute instead of, or as well as, the `NAME="..."` HTML tag attribute. With browsers converging on the DOM specification and its Nodal structure, the `NAME="..."` HTML tag attribute is expected to become deprecated in favor of the `ID="..."` attribute.

See also:	`Anchor.name`, `Document.<form_name>`, `ID="..."`, `Object.name`, `OBJECT.name`, `Plugin.name`, `Window.name`

NamedNodeMap object (Object/DOM)

Where nodes have a `name` attribute, they can be presented as members of a NamedNodeMap collection object.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0
Inherits from:	`Array` object
JavaScript syntax:	- *`myNamedNodeMap`* `= new NamedNodeMap()`
Object properties:	`length`
Object methods:	`getNamedItem(), item(), removeNamedItem(), setNamedItem()`

A `NamedNodeMap` is a collection of nodes that can be accessed by name. It does not inherit from `NodeList`, and the DOM does not mandate any parentage. It will probably inherit from `Collection` or `Array` but this appears to be implementation dependant. A general purpose `Dictionary` class would be helpful as a starting point but these are not available in all implementations.

The nodes are not collated in any particular order and you can access them with a numeric index as well as by associative name.

This implies a namespacing issue and clearly there may be problems with your document if nodes share the same name and need to be collected into a `NamedNodeList` entity.

The DOM level 2 specification provides these new methods to cope with namespaces:

- `getNamedItemNS()`
- `setNamedItemNS()`
- `removeNamedItemNS()`

DOM 2 standardization is not quite stable and no browsers support it yet. At this time, browsers have reached DOM level 1 capabilities and we can expect support for these namespace extensions in the next round of browser upgrades.

See also:	`Collection` object, DOM, DOM –Level 1, `Node` object, `Node.attributes[]`, `NodeList` object

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`length`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`getNamedItem``()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`item()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	Warning
`removeNamedItem()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`setNamedItem()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Inheritance chain:

`Array` Object

Web-references:

http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html#ID-1074577549

NamedNodeMap.getNamedItem() (Method)

Given the name of a node, it can be extracted from the collection by name.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*`myNamedNodeMap`*`.getNamedItem(`*`aName`*`)`
Argument list:	*`aName`*	The name of the node to be accessed

If a node having the specified name exists in the node map, the a reference to it will be returned. If there is no matching node, a `null` will be returned instead.

It isn't clear what happens if there are duplicate nodes with the same name. Some implementations may choose to return a collection of all nodes with a matching name.

NamedNodeMap.item() (Method)

The usual collection access by item number also applies to named node maps.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNamedNodeMap*`.item(`*anIndex*`)`
Argument list:	*anIndex*	A reference to an element in a collection

This accesses nodes by their position in the linear sequence of nodes within the map. The index value is zero-based so if the value specified is equal to or greater than the number of nodes in the map, a null value will be returned instead.

Warnings:

- Note that the `item()` method for a `NamedNodeMap` is spelled with a lower case i. The MSIE `Collection` class also has an `Item()` method that provides similar (but not identical) ways of accessing items in a collection. Note that `Collection.Item()` is spelled with a capital I.

See also:	`Collection.Item()`

NamedNodeMap.length (Property)

The number of members in the named node map is returned by this property.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myNamedNodeMap*`.length`

This is zero-based length of the NamedNodeMap collection/array. It allows you to enumerate through the collection with a `for(...)` loop, visiting each node in turn.

NamedNodeMap.removeNamedItem() (Method)

Given that you know the name of an item, you can locate and remove it from the collection. If necessary, the item is replaced by another containing the default attribute settings.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNamedNodeMap*`.removeNamedItem(`*aName*`)`
Argument list:	*aName*	An associative array reference

The value returned is a reference to the node that was removed. The same rules used for the *getNamedItem*`()` method apply. If no item is found then a null is returned instead. Multiple matching nodes may result in unpredictable and implementation-specific results.

If this node is not referred to by any other means and you do not assign the value to a variable, then the node will become detached and no longer has an owner. You will have no way to locate that node again without reconstructing the document. The object representing the node should in due course be garbage-collected automatically.

NamedNodeMap.setNamedItem() (Method)

A node is added to the collection having the specified node name. Any node already present with that name will be replaced.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNamedNodeMap*`.setNamedItem(`*aNode*`)`
Argument list:	*aNode*	The node to be set

If a replacement happens, the old node that occupied the same position in the document and which was displaced by the new node will be returned as a result of this method call.

If you need to access that node again, you must make sure a reference to it is retained otherwise you will need to reconstruct the document from scratch to manufacture another.

Namespace (Definition)

A table where identifiers are stored.

A namespace behaves like a dictionary table where lexically sorted items can be stored. There may be several namespaces. Each namespace is distinct from any other and so a particular value may be present in more than one namespace and may mean a different thing in each.

For example, the collection of properties belonging to an object class is referenced via a namespace. Each class maintains a separate namespace and so a property with a particular name can be added to several objects without any collisions.

The namespace where variable names and functions belong is also the same namespace where reserved words and keywords are accessed. Or at least this is the conceptual idea, when the standard recommends that you should not name your identifiers with the same value as a reserved word.

A namespace could be conceived as a particular directory within a file system or a set of object ID names within a document model. All of these operate with typical namespace behaviors regarding collisions between dissimilar objects having the same name.

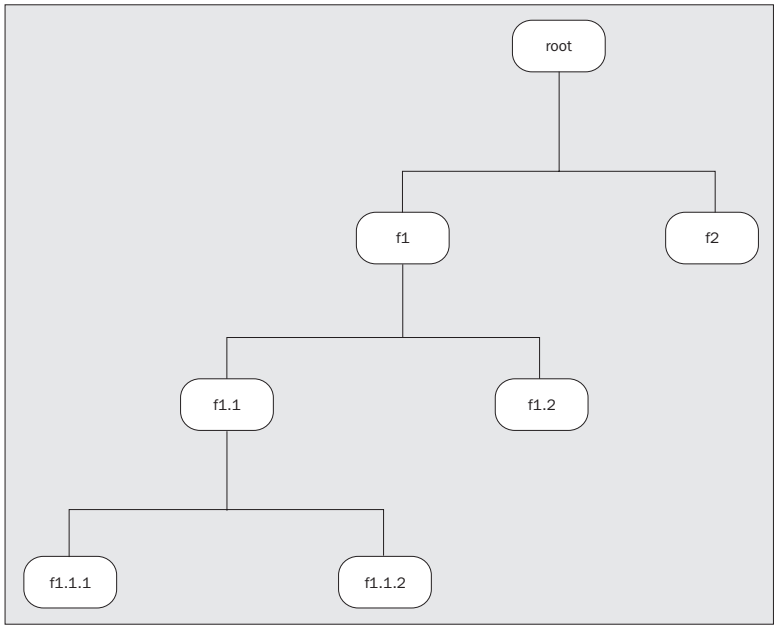
Namespace pollution can be alleviated by prefixing the names of certain kinds of objects. You might have two functions that create named items in a namespace and to ensure they do not operate on each other's entities, they could each add a different prefixing letter to the entity names.

Style sheets use namespaces as well to indicate different functionality during the cascading process.

The scope-chain rules of execution contexts within a JavaScript allow for namespaces to be expanded and collapsed as functions are called and executed. Variables can be created with a scope that is local to within a specific function. However the scope-chain mechanism allows the namespace to be layered and cascaded so that a reference to an identifier that is not local to the function but exists higher up the scope chain is still reachable. This means that the namespace for variables in JavaScript operates in a tree-like manner.

You may find in some implementations that there are small variations in the namespace behavior. Historically, language developers have found this to be a contentious area. In particular, there was a stage during the development of the C language standard where members of all structures were considered to live within the same namespace. Fortunately, the standard rejects this model and members of structures live in separate namespaces, one per structure. So it should be for objects in JavaScript, given the caveat that prototype inheritance can cloud the issue and a prototype tree for objects operates essentially in the same way as the scope chain does for functions.

Typically the object, method and property names are spelled inconsistently for some items. There are wide spread inconsistencies in the JavaScript language implementations. Although the core language is specified reasonably consistently, the browser manufacturers have added a large number of extensions that do not follow consistent naming conventions.



See also:	Identifier, Member, `OBJECT.name`, Prototype Based Inheritance, Scope chain

NaN (Constant/static)

A literal constant whose type is a built-in primitive value.

Availability:	ECMAScript edition –2 JavaScript –1.3 JScript –2.0 Internet Explorer –4.0 Netscape –4.06 Netscape Enterprise Server –2.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- `NaN`

The primitive value `NaN` represents the IEEE 754 "Not-a-Number" value. This is returned when the result of an evaluation is known to yield a numeric value but its magnitude and sign is uncertain. Because the value is numeric but uncertain, you cannot compare `NaN` with anything else (including itself). However you can test for its existence with the `isNaN()` function.

In IEEE 754, there are many millions of possible values for `NaN`. In ECMA-compliant JavaScript interpreters, they are all collected together and referred to as a single value. This means you cannot distinguish the reason for the `NaN` error as you may be able to in other languages that use IEEE 754 arithmetic.

It is possible in the hosting environment to provide additional facilities to determine what sort of `NaN` values you have. This is implementation dependant however and not part of the standard.

If you are in an environment that does not have the `NaN` value implemented, then you may be able to create one yourself like this:

```
var NaN = 0/0;
```

The value has existed since JavaScript 1.1 but it was given a property name in JavaScript 1.3. Therefore from scripting point of view, its availability is defined as JavaScript 1.3 and not 1.1.

Warnings:

- Be careful not to assign your own values to this variable. You can corrupt it in some implementations. In MSIE version 5 for Macintosh, assigning a value to the global `NaN` value changes its setting but leaves `Number.NaN` unaffected. You cannot modify `Number.NaN`.
- Version 3.02 of MSIE with JScript version 1.0 silently converts `NaN` values to zero. It does not know what `NaN` is.
- Netscape 2.02 cannot tell the difference between `null` and `undefined`.
- This constant is available as a property of the `Global` object in MSIE version 4 but not in Netscape 4.

See also:	Arithmetic constant, Exception, `Global` object, Global special variable, IEEE 754, Infinity, `isNaN()`, Not a Number, null, Number, `Number.NaN`, Range error, Special number values, `Value` property

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –4.3.23

ECMA 262 edition 2 –section –15.1.1.1

ECMA 262 edition 3 –section –4.3.23

ECMA 262 edition 3 –section –15.1.1.1

Wrox *Instant JavaScript* –page –14

native (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Native feature (Definition)

That which is supported in the base level of the language.

Native features are those that are defined in the ECMA-262 standard (ECMAScript). That is anything that is fundamentally and solely JavaScript.

Hooks to activate some piece of hardware or database loading facilities and document objects are not considered fundamental and are therefore not native features of the language.

Adding two numbers together, instantiating objects with a new operator, or concatenating strings are fundamental, even if the objects being instantiated are host objects. The functionality is native because the new operator is a native capability of the language. You can instantiate any object of any type with the new operator as long as there is a constructor function available.

However, some implementations may provide enhancements to the native features without them being classified as host environment capabilities. These might be additional operators or extra function in the `Math` object for example. Maybe you could do more sophisticated things with `String` objects. This all fits under the category of enhancements to the fundamental language.

See also:	ECMAScript, JavaScript language

Cross-references:

Wrox *Instant JavaScript* –page –12

Native object (Definition)

One of the built-in objects that the core implementation provides.

Availability:	ECMAScript edition –2

A native object is any object supplied by the interpreter that is not considered part of the hosting environment.

Some native objects are built into the core interpreter while others may be constructed by executing script code. These are sometimes called built-in objects.

Here is a list of native object types that all ECMA-compliant interpreters must support:

- `Array`
- `Boolean`
- `Date`
- `Function`
- `Math`
- `Number`
- `Object`
- `String`

The `Global` object is added to the scope chain of a program when it commences execution. Other built-in objects are accessible as initial properties of the `Global` object. Some of these are added as core functionality and are available in all implementations. Others are added as host objects defined differently for each implementation.

Many built-in objects are functions. They can be invoked with arguments. Some of these are constructors; most are associated with objects so that they can be used as methods. These are functions that are intended to be used with the new operator.

See also:	`Array` object, `Boolean` object, Built-in function, `Built-in` object, Cast operator, `Date` object, `Function` object, `Global` object, `Math` object, `Navigator.appVersion`, `Number` object, `Object` object, `prototype` property, `String` object

Cross-references:

ECMA 262 edition 2 –section –4.3.6

ECMA 262 edition 2 –section –8.6.2

ECMA 262 edition 2 –section –15

ECMA 262 edition 3 –section –4.3.6

ECMA 262 edition 3 –section –8.6.2

ECMA 262 edition 3 –section –15

navigate() (Method)

Load a new URL into the window.

Availability:	JScript –1.0 Internet Explorer –3.02	
Property/method value type:	`undefined`	
JavaScript syntax:	IE	*myWindow*`.navigate(`*aURL*`)`
	IE	`navigate(`*aURL*`)`
Argument list:	*aURL*	A new location to navigate the window to

Warnings:

- This is variously referred to as a `navigate()` method and a `Navigate()` method. Note the different capitalization.
- According to some (but not all) Microsoft documentation on this, it is also a shortcut to the `navigateFrame()` method. However, this is at variance with the coverage under DHTML, which does not mention `navigateFrame()` at all.

Refer to:

`Window.navigate()`

navigator (Property)

An alias for the `window.navigator` property.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0 Opera –3.0	
Property/method value type:	`Navigator` object	
JavaScript syntax:	-	*myWindow*`.navigator`
	-	`navigator`

See also:	Cross-platform compatibility, `Window.clientInformation`, `Window.navigator`

Property attributes:

`ReadOnly`.

Navigator object (Object/browser)

An object that contains properties that describe the browser (a.k.a. user agent or client).

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
JavaScript syntax:	-	*myWindow*`.navigator`
	-	`navigator`
Object properties:	`appCodeName, appMinorVersion, `*`appName`*`, ` *`appVersion`*`, browserLanguage, constructor, cookieEnabled, cpuClass, language, onLine, opsProfile, platform, securityPolicy, systemLanguage, userAgent, userLanguage, userProfile`	
Object methods:	`javaEnabled(), plugins, preference(), savePreferences(), taintEnabled()`	
Collections:	`mimeTypes[], plugins[]`	

The `navigator` object is named after Netscape Navigator but is also present in other browsers since it has become the defacto standard way of enquiring as to a browser's provenance.

You can inspect the various properties belonging to the `navigator` object and establish the name and type of the browser, its version, and the platform it is running on.

The `navigator` object is available as a property of the `window` and also the `Global` object but it will be the same `navigator` object. There is only one and you cannot instantiate another.

As this object is persistent, you might add properties to it that can be passed from window to window. However this will only work on Netscape, because MSIE seems to initialize a new `navigator` object for each window. MSIE does allow you to add properties to the `navigator` object but they are private to that window. Netscape shares them across windows. The example shows how to make this work.

You can accomplish the same thing in a cross-browser portable manner by using a frameset and storing properties in the top-level frameset's `Global` object. You can also (if security allows) write scripts to communicate between windows, in which case you may need to make sure the scripts are aware of multiple `document` objects as well as windows. This can get tricky and you may need to make sure the scripts run in the correct windows and return a value to a caller. This should ensure they run in the correct scope.

Netscape 6.0 provides a sidebar, which you should expect to be persistent. There's a similar sidebar in MSIE too. Netscape clearly intends you to script its sidebar but right now the implementation is a bit buggy. It might be too much to hope for, but digging into the internals of these suggests that RDF is involved, which may point to some commonality. Maybe we will be able to use the sidebar in the browsers as a repository for session storage, or persistent values that we can access from script, but it's not there and working yet.

Warnings:

- You cannot enumerate the properties of the `navigator` (`clientInformation`) object in MSIE version 4.5 for Macintosh. This may apply to other platform implementations of MSIE version 4.5 as well.
- The `navigator` object is much better supported in version 5.0 of MSIE for Macintosh, which was released in the Spring of 2000.
- The example below was found not to work on Netscape 6, due to bugs.

Example code:

```
<!-- Save this in navigator.html -->

<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
navigator.myNewProperty = "session global value";
open("navigator1.html");
</SCRIPT>
</BODY>
</HTML>
-----------------------------------------------------------------
```

```
<!-- Save this in  navigator1.html -->
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
document.write(navigator.myNewProperty);
</SCRIPT>
</BODY>
</HTML>
```

See also:	Cross platform compatibility, `Window.navigator`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`appCodeName`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	ReadOnly
`appMinorVersion`	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly
`appName`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	ReadOnly
`appVersion`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	ReadOnly
`browserLanguage`	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly
`constructor`	1.2 +	-	4.0 +	-	-	-	Warning
`cookieEnabled`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	ReadOnly
`cpuClass`	-	3.0 +	-	4.0 +	-	-	ReadOnly
`language`	1.2 +	-	4.0 +	-	5.0 +	-	Warning, ReadOnly
`onLine`	-	3.0 +	-	4.0 +	-	-	ReadOnly
`opsProfile`	-	5.0 +	-	5.0 +	-	-	Warning, ReadOnly
`platform`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	-	ReadOnly
`securityPolicy`	1.4 +	-	4.7 +	-	-	-	Warning, ReadOnly
`systemLanguage`	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly
`userAgent`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	Warning, ReadOnly
`userLanguage`	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly
`userProfile`	-	3.0 +	-	4.0 +	-	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
`javaEnabled()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-
`plugins`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	Warning
`preference()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	Warning
`savePreferences()`	1.2 +	-	4.0 +	-	-	-	Warning
`taintEnabled()`	1.1 +	3.0 +	3.0 +	4.0 +	5.0 +	-	Warning, Deprecated

Navigator.appCodeName (Property)

The codename for the browser.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`navigator.appCodeName`

This is generally used for internal purposes and is often just another name for the browser. Because these properties are mostly evolved rather than thought out logically, there is a great deal of duplication and ambiguity in the `navigator` object.

Most browsers appear to report that their code name is `'Mozilla'`.

Property attributes:

`ReadOnly`.

Navigator.appMinorVersion (Property)

A version value supported by MSIE.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`navigator.appMinorVersion`

This is the version code following the dot. Thus version 5.0 of MSIE would yield the value 0. At least that is the theory. MSIE version 4.5 for Macintosh also yields the value 0 for this property.

Warnings:

- Be wary of the value returned by this property on MSIE. It may not be consistent with what you expect. Bug fixes and patches are normally not reflected in this value although you can parse `appVersion` or `userAgent` for further information and in the MSIE browser you can also get access to script engine version and build number values.

See also:	`ScriptEngine(),ScriptEngineBuildVersion(), ScriptEngineMajorVersion(), ScriptEngineMinorVersion()`

Property attributes:

`ReadOnly`.

Navigator.appName (Property)

The generic type or model name of the web browser.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*navigator*.appName

This is the name of the web browser with no version numbering information. It is not necessarily the same as the application code name property.

Here are some example values:

- `Netscape`
- `Microsoft Internet Explorer`

Property attributes:

`ReadOnly`.

Navigator.appVersion (Property)

The version and release information for the browser.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*navigator*.appVersion

This is the full version string for the browser.

Here are some typical values:

- `4.7 (Macintosh; I; PPC)`
- `4.0 (compatible; MSIE 4.5; Macintosh; U; PPC)`
- `4.0 (compatible; MSIE 5.0; Macintosh; I; PPC)`

Note that Netscape provides a version number in the form X.XX while MSIE only describes to X.X accuracy.

See also:	`Native` object

Property attributes:

`ReadOnly`.

Navigator.browserLanguage (Property)

The national language variant of the browser.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`navigator.browserLanguage`

This is a value you can check to ensure the user is operating a suitably configured browser for your pages. Nationality settings can affect many aspects of the script behavior such as:

- Font character sets
- Currency symbols
- Sort collation sequences
- Searching and comparing of strings

Refer to the Language codes topic for a list of the available codes.

This property is available in MSIE only. For scripts running in the Netscape browser, you should use the `Navigator.language` property.

See also:	Language codes, `Navigator.language`

Property attributes:

`ReadOnly`.

Navigator.browserLanguage (Property)

The national language version supported by the MSIE browser.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`navigator.language`

The value stored in this property will be the international standard code for languages. This may not be the same as the values specified in the ISO 3166 standard but it should conform to the ISO 639 standard, which allocates two and three letter codes to countries.

Refer to the Language codes topic for a list of the available codes.

This is equivalent to the `Navigator.language` property in the Netscape browser.

Warnings:

- Beware that use of this property name for some objects describes a scripting language and not an international language variant.
- On MSIE, this value is returned by the `browserLanguage` property of the `Navigator` object.

See also:	`Element.lang`, Language codes, `Navigator.browserLanguage`, `Navigator.language`, `Navigator.systemLanguage`, `Navigator.userLanguage`

Property attributes:

`ReadOnly`.

Navigator.constructor (Property)

A constructor for creating new `navigator` objects.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Navigator` object	
JavaScript syntax:	N	`navigator.constructor`

Netscape supports a `constructor` property for the `Navigator` object. You won't find very many circumstances where you will need to create a new instance of the `Navigator` object.

This is another example of where the generic approach to creating objects that Netscape employs may be less optimal than the special individually-coded objects support that MSIE employs (even though it leads to a more bulky implementation).

Warnings:

- Because Netscape supplies a constructor for virtually every object type that it supports, there is a constructor for the `Navigator` object class. However, trying to access the `navigator.constructor` value is quite hard. There seems to be a bug in its implementation that prevents it being converted to a primitive. However it still responds to a request for its `name` or `prototype` properties.

- Since this object behaves in this strange way, building general-purpose scripts that access constructors is quite hard. During our research, several general purpose inspector scripts were written. One that examined constructors worked on most objects but failed on the `Navigator` object.
- This may be a scope-chain problem, because if you pass a `Navigator` object to a function in one of its arguments, then accessing the constructor of that argument may yield the source text of the function. This problem is manifested in Netscape 4.7 for Macintosh and may be extant on other platforms.
- This doesn't seem to affect whether the other properties that the `Navigator` object supports can be enumerated in a `for( ... in ... )` loop.

Navigator.cookieEnabled (Property)

A flag value indicating whether cookies are available or not.

Availability:	JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`navigator.cookieEnabled`

You can use this property to check whether the user has enabled cookie support. The content of this property will be the Boolean `true` or `false` value. This may affect the way that you treat the user when they access your site.

Property attributes:

`ReadOnly`.

Navigator.cpuClass (Property)

An indication of what sort of processor is in the hardware.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`navigator.cpuClass`

This property reports the following types of CPU:

- `x86`
- `PPC`

The `x86` value would be yielded on systems running on an Intel processor such as a Pentium or when running a web browser inside a PC emulator on a Macintosh. The `PPC` value would be returned when running the browser as a native application in the Macintosh.

Property attributes:

`ReadOnly`.

Navigator.javaEnabled() (Method)

A method that tells you whether Java support is enabled in the browser or not.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`navigator.javaEnabled()`

You can use this property to check whether the user has enabled Java applet support. The value returned by this method will be the Boolean `true` or `false` primitive. This may affect the way that you treat the user when they access your site.

Navigator.language (Property)

The national language version supported by the browser.

Availability:	JavaScript –1.2 Netscape –4.0 Opera –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`navigator.language`

The value stored in this property will be the international standard code for languages. This may not be the same as the values specified in the ISO 3166 standard but it should conform to the ISO 639 standard, which allocates two and three letter codes to countries.

Refer to the Language codes topic for a list of the available codes.

Warnings:

- Beware that use of this property name for some objects describes a scripting language and not an international language variant.
- On MSIE, this value is returned by the `browserLanguage` property of the `Navigator` object.

See also:	`Element.lang`, Language codes, `Navigator.browserLanguage`, `Navigator.systemLanguage`, `Navigator.userLanguage`

Property attributes:

`ReadOnly`.

Navigator.mimeTypes[] (Collection)

This property returns an array of supported MIME types.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
JavaScript syntax:	-	`navigator.mimeTypes`

The MIME type array is a sub-class of the built-in `Array` class. It is called a `MimeTypeArray`.

Here is a list of MIME types supported in recent versions of MSIE and Netscape:

Type	Description
application/mac-binhex40	Macintosh BinHex Archive
application/macbinary	MacBinary application
application/msword	Microsoft Word Document
application/pdf	Adobe Portable Document Format (PDF)
application/postscript	PostScript File
application/rtf	Rich Text Format File
application/sdp	Session Description Protocol
application/smil	SMIL Document
application/streamingmedia	Standard Streaming Metafile
application/vnd.ms-excel	Microsoft Excel Worksheet
application/vnd.ms-powerpoint	Microsoft PowerPoint Presentation
application/vnd.rn-realmedia	RealMedia File
application/vnd.rn-realplayer	RealPlayer File
application/vnd.rn-realsystem-rjs	RealSystem Skin
application/vnd.rn-realsystem-rmx	RealSystem Secure Media Clip
application/vnd.rn-rn_music_package	RealJukebox Music Package
application/vnd.rn-rsml	RealSystem ML File
application/wordperfect5.1	WordPerfect PC 5.1 Doc

Table continued on following page

Type	Description
application/x-compress	Unix Compressed (.z) Files
application/x-director	Shockwave
application/x-dvi	TeX DVI Document
application/x-javascript	JavaScript Program
application/x-macbinary	MacBinary File
application/x-rtsp	Real Time Streaming Protocol
application/x-sdp	Scalable Multicast
application/x-stuffit	Macintosh StuffIt Archive
application/x-tar	Unix TAR Archive
application/zip	ZIP Archives
audio/basic	ULAW/AU Audio
audio/mid	MIDI
audio/mp3	MP3 Audio
audio/mpeg	MPEG audio stream
audio/mpegurl	MP3 PlayLists (.m3u,.pls)
audio/mpg	MP3 Audio
audio/scpls	MP3 PlayLists (.m3u,.pls)
audio/vnd.qcelp	QCP Audio
audio/vnd.rn-realaudio	RealAudio Clip
audio/x-aiff	AIFF Audio
audio/x-mp3	MP3 Audio
audio/x-mpeg	MPEG audio stream
audio/x-mpegurl	MP3 PlayLists (.m3u,.pls)
audio/x-mpg	MP3 Audio
audio/x-pn-realaudio	RealPlayer File
audio/x-scpls	MP3 PlayLists (.m3u,.pls)
audio/x-wav	WAV Audio
image/gif	GIF Image
image/ief	IEF image
image/jpeg	JPEG Image
image/pict	PICT Image
image/png	PNG Image
image/tiff	TIFF Image
image/vnd.rn-realflash	RealFlash Clip
image/vnd.rn-realpix	RealPix Clip
image/x-pict	PICT Image
image/x-portable-bitmap	Portable Bitmap (PBM)

Table continued on following page

Type	Description
image/x-portable-graymap	Portable Graymap (PGM)
image/x-portable-pixmap	Portable Pixmap (PPM)
image/x-quicktime	QuickTime Image
image/x-rgb	SGI RGB Image
image/x-xbitmap	X Bitmap Image
image/x-xpixmap	X-Windows Pixmap
text/html	HTML Document
text/plain	Text File
text/vnd.rn-realtext	RealText Clip
video/mpeg	MPEG video/audio stream
video/msvideo	Microsoft Video
video/quicktime	QuickTime Movie
video/vnd.rn-realvideo	RealVideo Clip
video/x-msvideo	Microsoft Video

Some MIME types are only supported by Netscape:

Type	Description
application/fractals	Fractal Image Format
application/futuresplash	FutureSplash Player
application/gzip	application/gzip
application/java-archive	Java Archive
application/ms-powerpoint	application/ms-powerpoint
application/octet-stream	Binary Executable
application/pre-encrypted	Pre-encrypted Data
application/vnd.lotus-1-2-3	Lotus 123 Document
application/vnd.lotus-approach	Lotus Approach Document
application/vnd.lotus-freelance	Lotus Freelance Document
application/vnd.lotus-organizer	Lotus Organizer Document
application/vnd.lotus-screencam	Lotus ScreenCam Movie
application/vnd.lotus-wordpro	Lotus WordPro Document
application/vnd.ms-access	Microsoft Access Database
application/vnd.ms-schedule	Microsoft Schedule+ Application
application/x-authorware-map	Authorware
application/x-compressed	application/x-compressed
application/x-conference	application/x-conference
application/x-cpio	Unix CPIO Archive
application/x-csh	C Shell Program

Table continued on following page

Type	Description
application/x-excel	application/x-excel
application/x-fortezza-ckl	Compromised Key List
application/x-gtar	GNU Tape Archive
application/x-javascript-config	JavaScript Config
application/x-latex	LaTeX Document
application/x-ns-proxy-autoconfig	Proxy Auto-Config
application/x-perl	Perl Program
application/x-pkcs7-crl	Certificate Revocation List
application/x-pkcs7-mime	PKCS7 Encrypted Data
application/x-pkcs7-signature	PKCS7 Signature
application/x-sh	Bourne Shell Program
application/x-shar	Unix Shell Archive
application/x-shockwave-flash	Shockwave Flash
application/x-tcl	TCL Program
application/x-tex	TeX Document
application/x-texinfo	GNU TeXinfo Document
application/x-zip-compressed	Zip Compressed Data
audio/rmf	audio/rmf
audio/x-rmf	audio/x-rmf
image/x-cmu-raster	CMU Raster Image
image/x-MS-bmp	Windows Bitmap
image/x-photo-cd	PhotoCD Image
image/x-portable-anymap	PBM Image
image/x-xwindowdump	X Window Dump Image
Netscape/Source	Special file type
Netscape/Telnet	Netscape/Telnet
Netscape/tn3270	Netscape/tn3270
text/x-vcard	VCard
video/x-mpeg2	MPEG2 Video
video/x-qtc	video/x-qtc

These MIME types are supported only by MSIE:

Type	Description
application/applefile	AppleSingle file
application/AppleLink	AppleLink Package
application/ArcMac	PC ARChive

Table continued on following page

Type	Description
application/BBEdit	ML Source
application/binary	Application Binary Data
application/Canvas	Canvas Drawing
application/cdf	Channels
application/CodeWarrior	Java Class File
application/Compact_Pro	Compact Pro Archive
application/DeArj	ARJ Archive
application/DiskCopy	Apple DiskCopy Image
application/Envoy	Envoy Document
application/Excel	Lotus Spreadsheet r2.1
application/FileMaker_Pro	FileMaker Pro Database
application/FileMaker_Pro_3	FileMaker Pro Database
application/Finder	OpenType Font
application/FoxBase+	DBase Document
application/GraphicConverter	Animated NeoChrome
application/HexEdit	Untyped Binary Data
application/JPEGView	OS/2 Bitmap
application/MacAmp	MPEG-1 Layer 3
application/MacAnim_Viewer	DL Animation
application/MacBooz	Zoo Archive
application/MacLHA	LHArc Archive
application/macwriteii	MacWrite Document
application/Microsoft_Word	Word for Windows Template
application/MoviePlayer	DV Video
application/netcdf	Channels
application/oda	ODA Document
application/PageMaker	PageMaker 3 Document
application/PF_Encrypt	Private File
application/pgp-keys	PGP Key File
application/Photoshop	PhotoShop Document
application/PictureViewer	OS/2 Bitmap
application/PlayerPro	669 MOD Music
application/QuarkXpress	QuarkXpress Document
application/Replica	Replica Document
application/ResEdit	Resource File
application/self-extracting	Self-Extracting Archive
application/Self_Extracting_Archive	Self-Extracting Archive
application/SimpleText	Apple documentation file

Table continued on following page

Type	Description
application/SoftWindows	MS-DOS Executable
application/SoundApp	Amiga OctaMed music
application/SoundHack	IRCAM Sound
application/StuffIt	StuffIt Archive
application/StuffIt_Expander	PackIt Archive
application/SunTar	Unix BAR Archive
application/vnd.fdf	Forms Data Format
application/waf	Website Archive
application/WordPerfect	WordPerfect PC 4.2 Doc
application/x-cdf	Channels
application/x-gocserve	CompuServe Inbound Link to CIM 3.0
application/x-gzip	GZIP File
application/x-hdf	HDF Data File
application/x-netcdf	Channels
application/x-sgml	SGML Document
application/x-x509-ca-cert	Certificates
application/xml	HTML Document
audio/aiff	AIFF Audio
audio/midi	MIDI
audio/wav	WAV Audio
audio/x-midi	MIDI
audio/x-pn-realaudio-plugin	RealPlayer Plugin
image/x-bmp	Windows BMP Image
image/x-fits	Flexible Image Transport
image/x-macpaint	MacPaint Image
image/x-macpict	PICT Picture
image/x-pbm	Portable Bitmap
image/x-pgm	Portable Graymap
image/x-photoshop	Photoshop Image
image/x-png	PNG Image
image/x-ppm	Portable Pixmap
image/x-sgi	SGI Image
image/x-targa	Targa Truevision Image
image/x-tiff	TIFF Image
image/x-xbm	X-Windows Bitmap
image/x-xpm	X-Windows Pixmap
image/x-xwd	X-Windows Dump

Table continued on following page

Type	Description
image/xbitmap	X Bitmap Image
image/xbm	X Bitmap Image
message/external-body	URL Bookmark
text/cdf	Channels
text/css	Text File
text/javascript	Text File
text/Jscript	Text File
text/url	URL File
text/vbs	Text File
text/vbscript	Text File
text/x-cdf	Channels
text/xml	HTML Document
video/avi	Microsoft Video
video/flc	FLC Animation
video/x-mpeg	MPEG video/audio stream
x-world/x-3dmf	QuickDraw 3D File
x-world/x-vrml	VRML File

The example inspects the available MIME types. For each one, it will display these properties (note that only one is shown as an example):

```
type:audio/x-rmf

description:audio/x-rmf

suffixes:rmf

enabledPlugin:[object Plugin]
```

Warnings:

- In MSIE version 4.5 for Macintosh, this returns the `undefined` value and therefore you cannot establish the MIME types that the browser will support. In MSIE for other platforms a `null` value may be returned.
- In Netscape the array element property name values are the same as the type name. In MSIE version 5, the values are simply the numeric index. This may affect the way that you access the array. However, you can access the `type` property of each element in the array and if you check that, you should be able to build a portable script.
- MSIE and Netscape support different sets of MIME types. Some are supported on both and some only on one. They also differ somewhat in the textual description of the MIME type so it is really only safe to test the `type` property and ignore the `description` property.
- If you write an inspector script to dump the array values out to the screen, you may observe that some types do not have a descriptive name and in a couple of cases, a MIME-type exists with a name but no correctly defined type value. This may be version- and platform-specific. The values reported by this array may also be modified according to what you do in the preferences panels regarding the mapping of file types to helper applications.

- Many MIME types that are browser-specific may actually be similar to MIME types in other browsers. Because these are created and defined by different browser manufacturers there are always likely to be some browser-dependent differences.
- The associative array technique may not always work across browsers.

Example code:

```
<!-- Show all mime types installed -->
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
for (var myIndex=0; myIndex<navigator.mimeTypes.length; myIndex++)
{
   for (var myProperty in navigator.mimeTypes[myIndex])
   {
      document.write(myProperty);
      document.write(":");
      document.write(navigator.mimeTypes[myIndex][myProperty]);
      document.write("<BR>");
   }
   document.write("<HR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`MimeType` object, `MimeTypeArray` object

Property attributes:

`ReadOnly`.

Navigator.onLine (Property)

This property tells you whether the browser is online to a network.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`navigator.onLine`

If you select the work offline mode then this property will return `false`. It returns `true` when you go back online. This allows you to write scripts that will operate differently when they are being executed in an offline session.

Property attributes:

`ReadOnly`.

Navigator.opsProfile (Property)

An undocumented property of the `Navigator` object.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`Undefined`	
JavaScript syntax:	IE	`navigator.opsProfile`

This property is visible when the `Navigator` object is placed in a enumerating `for( ... in ... )` loop. However, there does not appear to be any documentation available on what it is used for. Its name suggests that it might be something to do with user preferences.

It seems to be available only on some versions and platforms. It shows up on MSIE version 5.0 for Macintosh but not on Windows.

Warnings:

- This yields the `undefined` value in MSIE and Netscape .

Property attributes:

`ReadOnly`.

Navigator.platform (Property)

The name of the hardware or operating system that the browser is running on.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`navigator.platform`

Using this property to optimize the page content for the platform can yield some benefits in performance at the expense of a little more work required to develop the site content.

Here are some possible values for this property:

- `Win16`
- `Win32`
- `MacPPC`
- `Mac68K`

Other values will be received by users of Unix systems.

This at least lets you distinguish between users running on one of the modern versions of Windows vs. a Macintosh user, or a user of an older platform. You can then opt to present simpler content with less client-side rendering effort on a lesser platform.

Property attributes:

`ReadOnly`.

Navigator.plugins.refresh() (Method)

Refresh all the plugins in the current page.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
JavaScript syntax:	-	`navigator.plugins.refresh()`

This is not strictly a direct method belonging to the `Navigator` object but it is helpful to remind you about it if you are looking at the `Navigator.plugins` property.

Calling this method is recommended if you suspect that a new plugin has been installed since the current browsing session was started. Actually, this is likely to happen quite rarely, but nevertheless, calling `Navigator.plugins.refresh()` as a matter of course at the start of your plugin access script will not cause any undue harm.

Navigator.plugins[] (Collection)

An array of the plugins currently installed into the browser.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
JavaScript syntax:	-	`navigator.plugins`

The object returned is a sub-class of the `Array` object. It is a `PluginArray`.

Here is a list of some plugins displayed by using JavaScript to unwrap the `plugins` array:

- Headspace Beatnik Helper Stub Plugin V1.0.1
- LiveAudio
- Shockwave for Director

- PDFViewer
- QuickTime Plug-in 4.1
- RealPlayer(tm) G2 LiveConnect-Enabled Plug-in (Mac)
- Shockwave Flash

The default plugin has been omitted since it is always there. However note that it is not always spelled the same in all browsers. If you are displaying a list of plugins to the user, then be careful how you eliminate unwanted items.

You may check for the existence of a plugin using the following logical expression:

```
(navigator.plugins["TheRequiredItem"] != null)
```

However, this may not always work due to the naming conventions.

The example enumerates some plugins and of course the output depends on the plugins you have installed. You might typically see something like this listed:

- Headspace Beatnik Helper Stub Plugin V1.0.1
- Default Plug-in
- LiveAudio
- Shockwave for Director
- PDFViewer
- QuickTime Plug-in 4.1
- RealPlayer(tm) G2 LiveConnect-Enabled Plug-in (Mac)
- Shockwave Flash

Warnings:

- This sometimes yields the `undefined` value and therefore it is impossible to determine which plugins are supported in MSIE version 4.5 for Macintosh and other browser/platform combinations that do not support the `plugins` array.
- If this is supported in a target browser you are writing scripts for, the exact results will depend on the user's configuration. There is enormous scope for users to install a variety of free and downloadable plugins from the web. There is no guarantee as to what plugins they will have installed. Their browser may only report a couple of plugins (theoretically the minimum is just the one default plugin) or they may have many plugins installed.
- The associative array technique may not always work across browsers.
- Netscape and MSIE encapsulate plugin/embedded objects in different ways. In MSIE they are objects of the `EMBED` class. In Netscape they are objects commonly referred to as belonging to the `Plugin` class although they are really implemented as `JavaObject` objects. In MSIE, this is an `ActiveX` object.

- There is additional confusion in that there is a `plugins[]` array that belongs to the document and another than belongs to the `navigator` object. They both contain collections of objects but of different types. This is further confused by the fact that the `document.plugins[]` array is another name for the `document.embeds[]` array.
- To help clarify this confusing situation, the best recommendation is that we refer to `document.embeds[]` and `navigator.plugins[]` and quietly ignore the `document.plugins[]` array. Furthermore we shall refer to `Plugin` objects as being something the browser can use to play embedded content and `Embed` objects will be an instance of a plugin that is alive and running in a document.

Example code:

```
<!-- Display all plugin names -->
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
for (var myIndex=0; myIndex<navigator.plugins.length; myIndex++)
{
   document.write(navigator.plugins[myIndex].name);
   document.write("<HR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Plugin` object, `PluginArray` object

Property attributes:

`ReadOnly`.

Navigator.preference() (Method)

A mechanism for allowing signed scripts to set a preference value in Netscape .

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
JavaScript syntax:	-	`navigator.preference()`
	-	`navigator.preference(`*`aName`*`)`
	-	`navigator.preference(`*`aName`*`, `*`aValue`*`)`
Argument list:	*`aName`*	The name of a preference
	`aValue`	A new value to store in a preference

This is not portable but may be necessary in some captive intranet situations where you know what the target client is going to be and have some control over it.

Although the method appears to be widely available, the values for specifying user preferences certainly have not been standardized and the security implications are also not common to all the platforms. You should be prepared to write some portability support code if you plan to deploy this functionality.

Warnings:

- You cannot access this method unless the script has the `UniversalPreferencesRead` privilege. Setting preferences requires the `UniversalPreferencesWrite` privilege.

See also:	`UniversalPreferencesRead`, `UniversalPreferencesWrite`

Web-references:

http://developer.netscape.com/library/documentation/deploymt/jsprefs.htm

Navigator.savePreferences() (Method)

Save the preference values for the current user.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	`navigator.savePreferences()`

Any access to the preferences mechanisms in a browser for any purpose other than reading them should be treated with some suspicion. It may be preferable to invite the user to adjust a preference setting rather than to modify it for them. You might affect the behavior of other pages inadvertently.

Warnings:

- In Netscape, your script needs to be granted the `UniversalPrefrencesWrite` privilege before this call can be executed successfully.

See also:	`UniversalPreferencesWrite`

Navigator.securityPolicy (Property)

An indication of the current security policy settings.

Availability:	JavaScript –1.4 Netscape –4.7	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	`navigator.securityPolicy`

This property contains the current security policy settings. Examining this property in a browser session yields the value "`Export policy`".

Warnings:

- This yields the `undefined` value in MSIE for Macintosh.

Property attributes:

`ReadOnly`.

Navigator.systemLanguage (Property)

An MSIE-specific language property.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`navigator.systemLanguage`

This property describes the system-specified language and country code. It uses the same language values as the `navigator.language` property.

Refer to the Language codes topic for a list of the available codes.

The most likely default value you will encounter will be "en" which signifies English but does not make distinction between UK and US variants. The value is supposed to be case-insensitive.

Warnings:

- Note that some platforms will yield a language and country code while others may only yield a language code.

See also:	`Element.lang`, ISO 3166, Language codes, `Navigator.language`, `Navigator.userLanguage`

Property attributes:

`ReadOnly`.

Navigator.taintEnabled() (Method)

This feature is no longer recommended. It is part of a now defunct security mechanism.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0 Deprecated	
JavaScript syntax:	-	`navigator.taintEnabled()`

This method is only supported in order to prevent scripts from crashing. The data-tainting support was a short-lived means of sending data back to a server. The security implications became unworkable and the whole data-tainting idea was deprecated.

The functionality was removed in JavaScript version 1.2. If you encounter this method in a script you are maintaining, you should seek to remove it to prevent run-time errors in the future.

If a browser implements this at all, it should return the value `false` for this method. Some earlier Netscape browsers may not. This functionality is highly deprecated and you can expect it to cause run-time exceptions in future.

Warnings:

- DO NOT USE THIS FACILITY!

See also:	`taint(),untaint()`

Navigator.userAgent (Property)

The identifying string for this browser.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`navigator.userAgent`

This string is returned to a web server in the `USER-AGENT:` header of an HTTP request. This is often logged and processed to analyze the distribution of browser types across the user base for a site.

The `userAgent` string contains much useful information and generally includes everything the `appVersion` and `appName` properties would have told you as well.

Here are some typical values:

- `Mozilla/4.7 (Macintosh; I; PPC)`
- `Mozilla/4.0 (compatible; MSIE 4.5; Mac_PowerPC)`
- `Mozilla/4.0 (compatible; MSIE 5.0; Mac_PowerPC)`

Warnings:

- Beware that some customized browsers will encode a variety of strange textual values into their `userAgent` property.

- While this will only affect the user of the browser when they run your script, the nature of the `userAgent` string may cause your script to fail.
- The following unusual values have been encountered in `userAgent` strings:
  - - Null characters
  - - Control characters
  - - URL strings
  - - HTML
  - - C-Code
  - - JavaScript
  - - GIF image data
  - - Very large text blocks
- These appear to be intended to cause server difficulties and possibly malfunctions in logging analysis systems. They may, as a side effect, cause your script to crash and burn too.

Example code:

```
// A function to normalize the user agent string
function getBrowserType()
{
   var myUserAgent;
   var myMajor;
   myUserAgent   = navigator.userAgent.toLowerCase();
   myMajor       = parseInt(navigator.appVersion);
   if( (myUserAgent.indexOf('mozilla')    != -1) &&
      (myUserAgent.indexOf('spoofer')    == -1) &&
      (myUserAgent.indexOf('compatible') == -1) &&
      (myUserAgent.indexOf('opera')      == -1) &&
      (myUserAgent.indexOf('webtv')      == -1))
   {
      if (myMajor > 4)
      {
         return "nav6";
      }
      if (myMajor > 3)
      {
         return "nav4";
      }
      return "nav";
   }
   if (myUserAgent.indexOf("msie") != -1)
   {
      if (myMajor > 4)
      {
          return "ie5";
      }
      if (myMajor > 3)
      {
         return "ie4";
      }
```

```
        return "ie";
    }
    return "other";
}

alert(getBrowserType());
```

See also:	`server.agent`

Property attributes:

ReadOnly.

Navigator.userLanguage (Property)

An MSIE specific language property.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`navigator.userLanguage`

This property describes the user-selected language code. It uses the same language values as the `navigator.language` property.

Refer to the Language codes topic for a list of the available codes.

Warnings:

- Note that some platforms will yield a language and country code while others may only yield a language code.

See also:	`Element.lang`, ISO 3166, Language codes, `Navigator.language`, `Navigator.systemLanguage`

Property attributes:

`ReadOnly`.

Navigator.userProfile (Property)

This yields a user profile object in MSIE browsers.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`UserProfile object`	
JavaScript syntax:	IE	`navigator.userProfile`

User profiles are available only on MSIE. The same arguments regarding uninvited modification and possible knock-on effects apply here just as much as they do to Netscape. Even though the scripting techniques may be different, the consequences may be similar.

See also:	`vCard` object

Property attributes:

`ReadOnly`.

Negation operator (-) (Operator/unary)

Negate an operand's value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`-anOperand`*
Argument list:	*`anOperand`*	A numeric value that can be negated

The operand is evaluated and converted to a numeric value. The result is negated.

A positive value becomes negative and a negative value becomes positive.

This is functionally equivalent to:

```
anOperand *= -1
```

Which is equivalent to:

```
anOperand = anOperand * -1
```

And also:

```
anOperand = 0 - anOperand
```

Although this is classified as a unary operator, its functionality is really that of an additive operator.

The associativity is from right to left.

Refer to the Operator Precedence topic for details of execution order.

See also:	Add (+), Additive expression, Additive operator, Associativity, Decrement value (--), Operator Precedence, Subtract (-), Unary operator

Cross-references:

ECMA 262 edition 2 –section –11.4.7

ECMA 262 edition 3 –section –11.4.7

NES (Product)

An abbreviation for Netscape Enterprise Server.

Refer to:

Netscape Enterprise Server

nethelp: URL (Request method)

A special URL access mode for Netscape help screens.

Availability:	JavaScript –1.2 Netscape –4.0

This is a special request method provided by the Netscape browser to gain access to local client-side help resources. The resources are loaded from files that live in a help folder within the folder containing the browser application. You can access the files with a text editor and study how they work. They are just plain HTML and JavaScript; however, they provide a quite helpful embedded help system. In a large enterprise where you control the deployment of the browsers, you may want to augment the browser help with some internal help pages. You might want to add details of your help desk and how to contact it for instance.

These special URLs are not present in MSIE although there will be some internal resources in that browser that may provide customization opportunities in a similar way. Indeed, Microsoft provides an administrator kit which you can use to customize the browser installation.

You may also be able to obtain admin tools from Netscape and Microsoft to carry out legitimate customizations on the browsers before deploying them throughout your organization.

This can be called from JavaScript by setting the `location.href` of a window to a valid `nethelp:` location.

Here are some examples:

URL	Description
nethelp:netscape/home:start_here	Front page of the help system
nethelp:netscape/collabra:HELP_SEC_CERTS_CRYPTOMODS	Security help
nethelp:netscape/collabra:HELP_SEC_CERTS_ISSUERS	Security help
nethelp:netscape/collabra:HELP_SEC_INFO	Security help
nethelp:netscape/collabra:HELP_SEC_PASS_UNSET	Security help
nethelp:netscape/collabra:HELP_SEC_PREFS_APPLET	Security help
nethelp:netscape/collabra:HELP_SEC_PREFS_MESSENGER	Security help
nethelp:netscape/collabra:HELP_SEC_PREFS_NAVIGATOR	Security help

Mostly, these special URLs will be useful for debugging. Getting details of the disk cache for example may be useful. Pulling up the JavaScript debugger page if you detect an error in your script might also be a cool trick.

This appears to have changed somewhat in Netscape 6.0, which is likely to provide a whole range of new possibilities. Once the browser is fully debugged and working we can spend many happy hours inspecting it to discover them.

Warnings:

- This is not present in MSIE at all. It displays help using a completely different mechanism altogether.

See also:	`about:` URL, `javascript:` URL, `mailbox:` URL, `mailto:` URL, URL

Netscape Enterprise Server (Product)

A web server that supports JavaScript on the server side.

Server-side scripting is accomplished by enclosing the JavaScript code inside `<SERVER>` tags in the HTML. These will be parsed out as the pages is processed and sent onwards. They will be replaced by the output of the script enclosed in the tags.

The main difference is that where you might have used a `document.write()` method in the client-side script, on the server side, you use a `write()` method.

If the `<SERVER>` tag remains intact and is forwarded in the document sent onwards to the browser, the browser should ignore it. However, this could give away much valuable information about the inside of the server and middle-ware. That could aid the wily hacker in an intrusion attack.

See also:	`blob` object, `client` object, `Connection` object, `Cursor` object, `database` object, `DbPool` object, `File` object, `Lock` object, `project` object, `request` object, `response` object, `ResultSet` object, `SendMail` object, `server` object, `Stproc` object

Cross-references:

Wrox *Instant JavaScript* –page –64

Netscape Navigator (Web browser)

One of the two major web browser platforms.

Netscape has made the source of the Communicator product available (which includes the Navigator browser). You can download this from the Netscape web site and participate in the development of the browser project. Netscape 6.0 is only just released for public use but is likely to contain some bugs.

We refer to this browser consistently as Netscape regardless of whether you are running Communicator or any other packaged version.

Here is a brief guide to versions of Netscape vs. JavaScript:

Version	JavaScript
2.0	JavaScript 1.0
3.0	JavaScript 1.1
4.0 to 4.05	JavaScript 1.2
4.06 to 4.75	JavaScript 1.3
5.0	JavaScript 1.4
6.0	JavaScript 1.5

Note that version 5.0 never shipped although if you search on the web you can find some installable binaries may satisfy your curiosity regarding browser history. Some reports suggest that those version 5.0 browsers only supported JavaScript 1.3. Because Mozilla is factored into components, it's feasible that you could build any version of the browser with whatever version of JavaScript you want.

Many values that Netscape exposes as JavaScript properties reflect the value of an HTML tag attribute. Likewise, many of its special objects are counterparts to the HTML tags. However it does not map objects so completely or consistently to HTML tags as the MSIE browser. Nor does it support the attributes and style mechanisms as elegantly.

Where the information is available, we have indicated the version number of JavaScript (or JScript) when HTML tags and attributes became accessible as objects or properties. In many cases, this may be a later version than when the instantiating HTML tag or attribute was first supported by the browser.

We constructed many scripts to inspect and enumerate the various properties of the objects in the MSIE and Netscape browsers. These uncovered many object types and properties that were hitherto undocumented. They might have been available in earlier versions of the browser. However, where language elements were discovered for the first time, they are initially documented as being available from version 4 of Netscape. A limited amount of further testing was applied where it was suspected that language elements may have been available in earlier releases and the availability modified accordingly.

Version 5 of Netscape was scrapped because its codebase became too unwieldy to work with. There seems little point in documenting its peculiarities. Netscape version 6.0 was in beta trials and until PR3 was so unstable and crash-prone that most of our testing bore little fruit. Right at the point where content was being finalized for publication Netscape 6.0 was released as a final product. It is clearly still a work in progress and there are quite a few non-working components. It looks good though. The potential for exercising the DOM standard document navigation is really exciting. There is a great deal yet to discover about the new browser and it will stabilize as bugs get fixed and new releases are shipped.

Perhaps browser versions may become less important as they converge on a single standard benchmark of functionality. For the time being, current practice suggests that version 4 browsers of both traditions are rapidly being taken over by version 5 MSIE browsers. Version 2 and 3 of MSIE and Netscape have declined to such small usage levels as to not require any further serious attempts to support them on new projects. Netscape 6.0 may win back some market share but only if its bugs are fixed quickly. Version 6.0 of MSIE is about to go to beta testers and the standards bodies are still some way ahead of the browser manufacturers so there is a long way to go yet.

Warnings:

- Netscape 2.02 does not cope gracefully with dollar signs in identifier names.

See also:	Identifier, Platform, Script execution, Undocumented features, Web browser

Cross-references:

Wrox *Instant JavaScript* –page –14

Web-references:

http://www.mozilla.org/ http://www.netscape.com/ http://home.netscape.com/browsers/6/index.html

netscape (Java package)

A short-cut reference to the `Packages.netscape` object. It allows JavaScript to access the Java class hierarchy to instantiate Java objects.

Availability:	JavaScript –1.1 Netscape –3.0 Netscape Enterprise Server –2.0	
Property/method value type:	`JavaPackage netscape`	
JavaScript syntax:	N	*myWindow*`.netscape`
	N	*myWindow*`.Packages.netscape`
	N	`netscape`
	N	`Packages.netscape`

See also:	`Window.java`, `Window.netscape`, `Window.Packages`, `Packages.netscape`

netscape.applet (Java package)

The root node of the Java hierarchy where the applets are built. A shortcut to the `Packages.netscape.applet` package.

Availability:	JavaScript –1.1 Netscape –3.0

This property returns a Java package that is encapsulated inside a JavaScript object belonging to the `JavaPackage` class.

You need to know quite a lot about the underlying Java objects to make use of this because the properties and methods it supports are not exposed in an enumerable fashion. This means you would have difficulty in writing a JavaScript inspector to examine these objects.

See also:	`JavaPackage` object

netscape.cfg (Java package)

A new style configuration file for Netscape. Not to be confused with the `Packages.netscape` Java classes.

Availability:	JavaScript –1.1 Netscape –3.0

See also:	`netscape.lck`, Preferences

netscape.javascript (Java package)

A Java package for supporting JavaScript inside Java. A shortcut to the `Packages.netscape.javascript` package.

Availability:	JavaScript –1.1 Netscape –3.0

Refer to:

`JavaPackage` object

netscape.javascript.JSObject (Java class)

The full definition of the `JSObject` class for encapsulating JavaScript objects in Java. A shortcut to the `Packages.netscape.javascript.JSObject` class.

Availability:	JavaScript –1.1 Netscape –3.0
Class methods:	`getWindow()`

Values of this type are visible to JavaScript as the encapsulated object that was originally passed to Java. Effectively, the wrapper is removed.

Note that you cannot use this as a constructor. There wouldn't be any point anyway because you can create JavaScript objects within the JavaScript environment; there is no need to go through the Java bridge to accomplish that.

See also:	Java calling JavaScript, Java to JavaScript values, JavaScript to Java values, `JSObject` object

netscape.lck (Java package)

A configuration file as used in older versions of Netscape. Not to be confused with `Packages.netscape`.

Availability:	JavaScript –1.1 Netscape –3.0

This is provided as support for preference management in older versions of Netscape.

See also:	Preferences

Cross-references:

Wrox *Instant JavaScript* –page –59

netscape.plugin (Java package)

The top of a hierarchy of Java packages that support plugins. This is a shortcut to the `Packages.netscape.plugin` package.

Availability:	JavaScript –1.1 Netscape –3.0

Refer to:

`JavaPackage` object

netscape.plugin.Plugin (Java class)

A special class for encapsulating plugins so they present a common API. This is a shortcut to the `Packages.netscape.plugin.Plugin` class.

Availability:	JavaScript –1.1 Netscape –3.0
HTML syntax:	`<EMBED>`
Object methods:	`destroy(), equals(), getClass(), getPeer(), getWindow(), hashCode(), init(), isActive(), notify(), notifyAll(), toString(), wait()`

See also:	`<EMBED>`, LiveConnect, `Plugin` object

Method	JavaScript	JScript	N	IE	Opera	Notes
`destroy()`	1.1 +	-	3.0 +	-	-	-
`equals()`	1.1 +	-	3.0 +	-	-	-
`getClass()`	1.1 +	-	3.0 +	-	-	-
`getPeer()`	1.1 +	-	3.0 +	-	-	-
`getWindow()`	1.1 +	-	3.0 +	-	-	-
`hashCode()`	1.1 +	-	3.0 +	-	-	-
`init()`	1.1 +	-	3.0 +	-	-	-
`isActive()`	1.1 +	-	3.0 +	-	-	-
`notify()`	1.1 +	-	3.0 +	-	-	-
`notifyAll()`	1.1 +	-	3.0 +	-	-	-
`toString()`	1.1 +	-	3.0 +	-	-	-
`wait()`	1.1 +	-	3.0 +	-	-	-

netscape.security (Java package)

The top of a hierarchy of Java packages that provide security facilities. This is a shortcut to the `Packages.netscape.security` package.

Availability:	JavaScript –1.1 Netscape –3.0

See also:	Security policy, `JavaPackage` object

netscape.security.PrivilegeManager (Java class)

Part of the Netscape security model implemented with Java.

Availability:	JavaScript –1.1 Netscape –3.0
Class properties:	`EQUAL, NO_SUBSET, PROPER_SUBSET, SIGNED_APPLET_DBNAME, TEMP_FILENAME, theDebugLevel`
Class methods:	`checkPrivilegeEnabled(), checkPrivilegeGranted(), disablePrivilege(), enablePrivilege(), enableTarget(), getMyPrincipals(), getPrivilegeManager(), getSystemPrincipal(), revertPrivilege()`
Object methods:	`disablePrivilege(), enablePrivilege(), getPrivilegeTableFromStack()`

Because the Netscape security model is based on the Java security model, the Netscape browser requests its privileges through the Java mechanisms. These are encapsulated in a class that you can access from inside JavaScript.

The downside of this is that there is no meaningful value returned when the request is made. If the request for a privilege is denied, the error causes a Java exception, which is difficult to trap from JavaScript. It is possible that more recent browser versions will support an exception-handling mechanism.

See also:	`PrivilegeManager` object, Requesting privileges, `UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`, `UniversalFileRead`, `UniversalPreferencesRead`, `UniversalPreferencesWrite`, `UniversalSendMail`

Method	JavaScript	JScript	N	IE	Opera	Notes
`disablePrivilege()`	1.1 +	-	3.0 +	-	-	-
`enablePrivilege()`	1.1 +	-	3.0 +	-	-	-
`getPrivilegeTableFromStack()`	1.1 +	-	3.0 +	-	-	-

new (Operator/unary)

An object construction operator.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	An object whose type depends on the constructor	
JavaScript syntax:	-	*myObject* = new *aConstructor*
	-	*myObject* = new *anObject*(*someArguments*)
Argument list:	*aConstructor*	An object constructor function
	anObject	An object to clone
	someArguments	A collection of initial values for the new instance

The new operator creates a new instance of the object it is operating on.

As the object is created, the receiver's Construct method is called with no arguments passed to it. Any initialization is only carried out by the Construct method.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

Typically this would be used to instantiate core objects of the following types:

- `Array`
- `Boolean`
- `Date`
- `Function`
- `Number`
- `Object`
- `RegExp`
- `String`

This can also be used to instantiate some host objects.

See also:	`Array()`, Associativity, `Boolean()`, `Date()`, `Function()`, Left-Hand-Side expression, List type, `Number()`, `Object()`, Operator Precedence, `RegExp()`, `String()`

Cross-references:

ECMA 262 edition 2 –section –11.2.2

ECMA 262 edition 2 –section –11.2.4

ECMA 262 edition 2 –section –15

ECMA 262 edition 3 –section –11.2.2

Wrox *Instant JavaScript* –page –15

Wrox *Instant JavaScript* –page –21

Newline (Escape sequence)

A means of introducing line breaks into string content texts.

Availability:	ECMAScript edition –2

The newline escape sequence `\n` can be placed in string literals if you want to break the output text over more than one line.

Warnings:

- Do not use a line terminator even with a backslash to escape it. In other words, don't think that placing a backslash as the last character of a line will hide the line terminator. While that may work in other languages, it won't work in JavaScript.

See also:	Escape sequence (\), Line terminator

Cross-references:

ECMA 262 edition 2 –section –7.2

ECMA 262 edition 2 –section –7.7.4

ECMA 262 edition 3 –section –7.3

Newlines are not <BR> tags (Pitfall)

A newline in a script does not display a line break in HTML output.

Because HTML is fairly freely formatted, you have to explicitly tell it when a line break is to appear. You can do this a variety of block-level tags. Its most likely done with a `<BR>` or `<P>` tag.

When you use the `document.write()` method, you need to explicitly include the necessary `<BR>` or `<P>` tags otherwise the HTML output that gets displayed will all run onto a single line. Placing a `\n` newline escape into the output may make the HTML source look nice but it won't affect the displayed output.

A line break is also introduced when you use block structured elements in the HTML. Here is a list of common HTML tags that do this (there are others too):

- <BLOCKQUOTE>
- <BODY>
-
- <DD>
- <DL>
- <DIV>
- <DT>
- <H1> etc
- <HR>
- <HTML>
- <LI>
- <OBJECT>
- <OL>
- <P>
- <PRE>
- <UL>

See also:	Pitfalls

Cross-references:

Wrox *Instant JavaScript* –page –46

News posts containing JavaScript (Advice)

You can embed JavaScript into news postings composed using HTML.

When you compose and post a news message, you may use HTML as a way to improve the presentation. This means you can include some JavaScript to be executed in the client mail-reader application.

Not all news-reading clients can support HTML let alone JavaScript. However, if your recipient does (and that is likely if its a web browser), then you can do some creative things to link topics in threads and pre-format replies so that they associate correctly into the news topic tree. You simply construct your HTML document in the normal way.

Warnings:

- There are significant security and virus related risks with JavaScript enabled news. The possibilities are so catastrophic that the best reccomendation is to deactivate JavaScript and Java in any news-reading client application.
- Just because something is possible does not mean it is advisable or good to do.
- On the other hand, within the confines of a closely controlled intranet or workgroup, this could find many useful applications. Just so long as you know where the news posts came from and you can absolutely trust that they have not been compromised.
- Personally, I'd recommend that you turn it off. It might be useful for including the odd diagram but mostly you need to get the informational content and it doesn't really need to be very pretty.

See also:	E-mail containing JavaScript

Cross-references:

Wrox *Instant JavaScript* –page –60

news: URL (Request method)

A request from a web browser to a news server to send a document.

Use the browser to download and browse some news content.

Warnings:

- This is only allowed under script control if the script has the `UniversalSendMail` privilege.

See also:	`javascript:` URL, `UniversalSendMail`, URL

Node object (Object/DOM)

A node is the primary component from which documents are built (in the context of a DOM hierarchy).

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
JavaScript syntax:	-	*myNode* = *myMutationEvent*.relatedNode
Object properties:	`firstChild, lastChild, nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, previousSibling`	

Class constants:	`ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE`
Object methods:	`appendChild(), cloneNode(), hasChildNodes(), insertBefore(), removeChild(), replaceChild()`
Collections:	`attributes[], childNodes[]`

Here is a list of the available node types:

Constant	Type	Description
`undefined`	null	A member of the attributes collection
`ELEMENT_NODE`	1	HTML element object node
`ATTRIBUTE_NODE`	2	HTML tag attribute object
`TEXT_NODE`	3	Text object node
`CDATA_SECTION_NODE`	4	CDATA section
`ENTITY_REFERENCE_NODE`	5	Entity reference
`ENTITY_NODE`	6	Entity node
`PROCESSING_INSTRUCTION_NODE`	7	Processing instruction node
`COMMENT_NODE`	8	Comment node
`DOCUMENT_NODE`	9	Document object
`DOCUMENT_TYPE_NODE`	10	Doctype object
`DOCUMENT_FRAGMENT_NODE`	11	Document fragment node
`NOTATION_NODE`	12	Notation node

The DOM level 2 specification adds the following methods:

- `supports()`
- `normalize()`

It also adds the following properties:

- `namespaceURI`
- `prefix`
- `localName`

At DOM level 3, the interface to the `Node` object is expected to evolve further to allow nodes to be compared and to be able to extract a serialized version of a DOM tree branch into a string primitive. This functionality is still under review as this is being written. The following additional properties are expected to be supported:

- `baseURI`
- `textContent`
- `key`

DOM level 3 is expected to add the following methods to the `Node` object:

- `compareDocumentOrder()`
- `compareTreePosition()`
- `isSameNode()`
- `lookupNamespacePrefix()`
- `lookupNamespaceURI`
- `normalizeNS()`
- `setUserData()`
- `getUserData()`

If the implementation supports the DOM level 2 event model (this is the case with Netscape 6.0), then the `Node` object should also support the methods and properties defined by the `EventTarget` object.

See also:	`Attribute.nodeType`, `Document` object, `DocumentFragment` object, `Element` object, `EventTarget` object, `MutationEvent.initMutationEvent()`, `MutationEvent.relatedNode`, `NamedNodeMap` object, `Node.firstChild`, `Node.insertBefore()`, `Node.lastChild`, `Node.nextSibling`, `Node.parentNode`, `Node.previousSibling`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`firstChild`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`lastChild`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`nextSibling`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`nodeName`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`nodeType`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`nodeValue`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`ownerDocument`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`parentNode`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`previousSibling`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`appendChild()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`cloneNode()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`hasChildNodes()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`insertBefore()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`removeChild()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`replaceChild()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Node.appendChild() (Method)

A new child node object is added to the end of the list of immediate children of this node.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.appendChild(`*aNode*`)`
Argument list:	*aNode*	The node to be appended

Node.attributes[] (Collection)

A `NamedNodeMap` collection of attributes for this node.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`NamedNodeMap` object	
JavaScript syntax:	-	*myNode*`.attributes`
See also:	`Element.attributes[]`, `MutationEvent.attrChange`, `MutationEvent.attrName`, `NamedNodeMap` object	

Node.childNodes[] (Collection)

A `NodeList` containing the immediate children of this node.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`NodeList` object	
JavaScript syntax:	-	*myNode*`.childNodes`
See also:	`Element.childNodes[]`, `NodeList` object	

Node.cloneNode() (Method)

The `node` object is cloned but the new instance has no parent node defined.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.cloneNode(`*aSwitch*`)`
Argument list:	*aSwitch*	Indicates whether a deep or shallow clone is required

Node.firstChild (Property)

The first object in the collection of direct children of this element.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.firstChild`
See also:	`Element.firstChild`, `Node` object	

Node.hasChildNodes() (Method)

A convenience method to provide a flag indicating whether there are any children belonging to this node.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myNode*`.hasChildNodes()`
See also:	`Element.contains()`, `Element.firstChild`	

Node.insertBefore() (Method)

This method inserts a child element into the collection at the indicated position.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.insertBefore(`*aNode1, aNode2*`)`
Argument list:	*aNode1*	The node to be inserted
	aNode2	The node indicating the insertion point

See also:	`Element.insertAdjacentText()`, `Node` object

Node.lastChild (Property)

The last child object contained within the DOM hierarchy that descends from this element.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.lastChild`

See also:	`Element.lastChild`, `Node` object

Node.nextSibling (Property)

An HTML element at the same level within the document hierarchy.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.nextSibling`

See also:	`Element.nextSibling`, `Node` object

Node.nodeName (Property)

The name value for this `node` object. This is provided by the HTML tag attribute.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNode*`.nodeName`

See also:	`Element.nodeName`

Node.nodeType (Property)

The nodes are created for a variety of purposes. This property indicates what sort of node has been instantiated.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myNode*`.nodeType`

The values in this parameter correspond to the values of the Static Constants belonging to the Node Class.

Here is a list of the available node types:

Constant	Type	Description
`undefined`	null	A member of the attributes collection
`ELEMENT_NODE`	1	HTML element object node
`ATTRIBUTE_NODE`	2	HTML tag attribute object
`TEXT_NODE`	3	Text object node
`CDATA_SECTION_NODE`	4	CDATA section
`ENTITY_REFERENCE_NODE`	5	Entity reference
`ENTITY_NODE`	6	Entity node

Table continued on following page

Constant	Type	Description
PROCESSING_INSTRUCTION_NODE	7	Processing instruction node
COMMENT_NODE	8	Comment node
DOCUMENT_NODE	9	Document object
DOCUMENT_TYPE_NODE	10	Doctype object
DOCUMENT_FRAGMENT_NODE	11	Document fragment node
NOTATION_NODE	12	Notation node

See also:	Element.nodeType

Node.nodeValue (Property)

The value of the node depends on the kind of node that the object encapsulates. Some nodes have `null` values.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myNode*.nodeValue

See also:	Element.nodeValue

Node.ownerDocument (Property)

The document that the node belongs to can be accessed via this property.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	Document object	
JavaScript syntax:	-	*myNode*.ownerDocument

The `Node.ownerDocument` property is similar to the `document` property of an `Element` object. In this case, because we are considering only DOM nodes, whether the document is HTML or not is of no consequence. For now it probably is because the work centers around browser capabilities, but the DOM specification is far more than just a better way to describe a web page.

They have very cleverly separated the description of an abstract document and the 'HTMLness' of it into two layers. That way, implementors can superimpose other types of document content on top of a generic DOM foundation built on the Node structure.

This property is effectively provided by MSIE version 5 as a property of an `Element` object. Netscape 6.0 makes it available here in a more DOM-compliant manner.

It is a convenience property because accessing a containing `document` object in this way is far easier than walking down a `document.childNode` sequence to locate a contained document.

See also:	`Document` object, `Element.ownerDocument`

Node.parentNode (Property)

The direct parent node of the current object. This object will be a member of the `childNodes` collection for that object.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.parentNode`

See also:	`Element.parentNode`, `MutationEvent.relatedNode`, `Node` object

Node.previousSibling (Property)

The object immediately before this one in the `childNodes` collection of their joint parent node.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.previousSibling`

See also:	`Element.previousSibling`, `Node` object

Node.removeChild() (Method)

A method for removing child nodes from the collection.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.removeChild(`*aNode*`)`
Argument list:	*aNode*	The node object to be removed

Node.replaceChild() (Method)

A means of replacing child objects with new nodes and discarding the old ones.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	*myNode*`.replaceChild(`*newNode*`, `*oldNode*`)`
Argument list:	*newNode*	The node to be placed into the hierarchy
	oldNode	The node to be replaced

NodeList object (Object/DOM)

A generic collection of nodes, not presented in any particular order.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
JavaScript syntax:	-	*myNodeList* `= ` *myDocument*`.getElements ByTagName(`*aTagName*`)`
Argument list:	*aTagName*	The name of an HTML tag
Object properties:	`length`	
Object methods:	`item()`	

This object is based on an `Array` object and contains a set of `Node` items contained within a DOM hierarchy.

Do not confuse DOM `NodeList` arrays with `Enumerator` or `Collection` objects. The `NodeListitem()` method is subtly different to the `Enumerator.Item()` method.

See also:	`Collection` object, `Document.getElementsByTagName()`, `Element.getElementsByTagName()`, `Enumerator` object, `NamedNodeMap` object, `Node.childNodes[]`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`length`	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	1 +	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`item()`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

NodeList.item() (Method)

An item extractor to retrieve a `Node` from within a `NodeList` collection.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Node` object	
JavaScript syntax:	-	`myNodeList.item(anIndex)`
Argument list:	*`anIndex`*	A numeric index to a `Node` in the collection

You should node that in MSIE, the `Collection` object has an `Item()` method whereas DOM mandates that a `NodeList` should have an `item()` method. Note the difference in the case of the initial letter.

Even though an implementation may forgive an upper-lower case error when it parses the script source text, an `item()` call on `NodeList` can only return a `Node` object referenced using an integer index location within the collection.

Because the `NodeList` is based on an `Array`, `NodeList.item(`*`anIndex`*`)` is functionally equivalent to `NodeList[`*`anIndex`*`]` and one can enumerate through the set of `Node` objects using either technique.

NodeList.length (Property)

Returns the length of a collection array.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myNodeList*.length

The `NodeList` collection behaves exactly like an `Array` object and returns a number representing a count of all the `Node` objects in the collection.

Example code:

```
myCount = document.all.length;
```

See also:	`Array.length`, `Collection.length`

Property attributes:

`ReadOnly`.

NOFRAMES object (Object/HTML)

An object that represents the content of a `<NOFRAMES>` tag.

Availability:	JScript –3.0 Internet Explorer –4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myNOFRAMES* = *myDocument*.all.*anElementID*
	IE	*myNOFRAMES* = *myDocument*.all.tags("NOFRAMES")[*anIndex*]
	IE	*myNOFRAMES* = *myDocument*.all[*aName*]
	-	*myNOFRAMES* = *myDocument*.getElementById(*anElementID*)
	-	*myNOFRAMES* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myNOFRAMES* = *myDocument*.getElementsByTagName("NOFRAMES")[anIndex]

HTML syntax:	`<NOFRAMES> ... </NOFRAMES>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`accessKey, tabIndex, dir`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

This was added to the MSIE browser at revision 4. It is a sub-class of the basic `Element` object and therefore shares many properties with other objects in the MSIE browser. Although it is implemented as an `Element`, it is not a DOM specified item.

See also:	`Element` object, `NOSCRIPT` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`accessKey`	-	3.0 +	-	4.0 +	-	-	-	-
`tabIndex`	-	3.0 +	-	4.0 +	-	-	-	-
`dir`	-	5.0 +	-	5.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onDblClick`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

NOFRAMES.dir (Property)

The direction of rendering of text contained within the block owned by the `<NOFRAMES>` tag.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myNOFRAMES*`.dir`

The `dir` property may be set to indicate a left-to-right or right-to-left parsing direction.

This is part of the localization support and represents the contents of the `DIR="..."` tag attribute.

If you assign a value to this property it is case-sensitive and must be either "`ltr`" or "`rtl`". Note that this is at variance with some documentation which says it is case-insensitive.

This property works in conjunction with the `lang` property to control the direction of text flow.

See also:	`BDO.dir,Element.dir,NOSCRIPT.dir`

Nombas ScriptEase (Product)

A standalone JavaScript interpreter.

This is an example of a JavaScript interpreter than can be used in several contexts.

The ScriptEase Web Server Edition (SEWSE) can be used in the backend of a server to accomplish much of what you might do with `<SERVER>` tags in the Netscape Enterprise Server or with Microsoft ASP. SEWSE is called as part of the CGI mechanism in the web server.

The Script Ease Desk Top implementation provides a way to use JavaScript to control desktop actions. This is available cross-platform and accomplishes the kind of things you might do with AppleScript on a Macintosh or Windows Script Host in the Microsoft Windows environment.

The SEWSE interpreter can also be used for general purpose scripting in the Unix environment in much the way you would use Bourne, Korn or C shells or the Perl interpreter.

You can also embed Script Ease into your own applications and add scriptability to them.

See also:	Standalone JavaScript

Nondigit (Definition)

A non-digit letter that can be used in an identifier.

In the context of creating identifier names, a non-digit character is a member of the following set:

```
a b c d e f g h i j k l m

n o p q r s t u v w x y z

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z
```

_ (Underscore)

See also:	Digit, Identifier

<NOSCRIPT> (HTML Tag)

A special tag that allows browsers that don't support scripting to display alternative content.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0

Browsers that support the `<SCRIPT>` tag should also support the `<NOSCRIPT>` tag. However, their support for the `<NOSCRIPT>` tag should be to hide or ignore anything between the `<NOSCRIPT>` tag and its corresponding `</NOSCRIPT>` closure.

Warnings:

- This is not supported by Netscape 2.

See also:	`<SCRIPT>`, Compatibility

NOSCRIPT object (Object/HTML)

An object representing the `<NOSCRIPT>` tag.

Availability:	JScript –3.0 Internet Explorer –4.0
Inherits from:	`Element` object

JavaScript syntax:	IE	*myNOSCRIPT* = *myDocument*.all.*anElementID*
	IE	*myNOSCRIPT* = *myDocument*.all.tags("NOSCRIPT")[*anIndex*]
	IE	*myNOSCRIPT* = *myDocument*.all[*aName*]
	-	*myNOSCRIPT* = *myDocument*.getElementById(*anElementID*)
	-	*myNOSCRIPT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myNOSCRIPT* = *myDocument*.getElementsByTagName("NOSCRIPT")[*anIndex*]
HTML syntax:	<NOSCRIPT> ... </NOSCRIPT>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	accessKey, tabIndex, dir	
Event handlers:	onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp	

This was added to the MSIE browser at revision 4. It is a sub-class of the basic Element object and therefore shares many properties with other objects in the MSIE browser. Although it is implemented as an Element, it is not a DOM-specified item.

See also:	Element object, NOFRAMES object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	-	3.0 +	-	4.0 +	-	-	-	-
tabIndex	-	3.0 +	-	4.0 +	-	-	-	-
dir	-	5.0 +	-	5.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

NOSCRIPT.dir (Property)

The direction of rendering of text contained within the block owned by the `<NOSCRIPT>` tag.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myNOSCRIPT*`.dir`

The `dir` property may be set to indicate a left-to-right or right-to-left parsing direction.

This is part of the localization support and represents the contents of the `DIR="..."` tag attribute.

If you assign a value to this property it is case-sensitive and must be either "`ltr`" or "`rtl`". Note that this is at variance with some documentation which says it is case-insensitive.

This property works in conjunction with the `lang` property to control the direction of text flow.

See also:	`BDO.dir`, `Element.dir`, `NOFRAMES.dir`

Not a number (Definition)

A special numeric value used to handle computational error conditions.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

IEEE 754 describes a numeric regime that includes the ability to flag exceptions when calculations generate invalid results. The result of such an expression may be signified by the `NaN` or Not a Number exception.

See also:	IEEE 754, `NaN`

Cross-references:

ECMA 262 edition 2 –section –4.3.23

ECMA 262 edition 3 –section –4.3.23

NOT Equal to (!=) (Operator/equality)

Compare two operands for inequality.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*anOperand1* `!=` *anOperand2*
Argument list:	*anOperand1*	A value to be compared
	anOperand2	Another value to be compared

The two operands are compared, and the Boolean `true` value is returned if they are not equal, otherwise `false` if they are equal. Note that JavaScript will attempt to convert both operands to the same type for comparison.

When testing for inequality, the following rule is invariant:

```
A != B
```

is equivalent to:

```
!(A == B)
```

Also, the rule of positioning allows that:

```
A == B
```

is identical to:

```
B == A
```

(Apart from the fact that exchanging the operands in this way alters the order in which they are evaluated.)

Exchanging the operands may have undesirable side effects if they are expressions. For example, they may call functions and test the results. If the functions are not totally independent of one another, you may get unexpected results.

The associativity is from left to right.

Refer to the Operator Precedence topic for details of execution order.

Refer to the Equality expression topic for a discussion on the ECMA standard definition of the equality testing rules.

See also:	ASCII, Associativity, Equal to (==), Equality expression, Equality operator, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), Logical expression, Logical NOT –complement (`!`), Logical operator, NOT Identically equal to (`!==`), Operator Precedence, Relational expression, Relational operator, typeof, Unicode

Cross-references:

ECMA 262 edition 2 –section –11.9.2

ECMA 262 edition 2 –section –11.9.3

ECMA 262 edition 3 –section –11.9.2

ECMA 262 edition 3 –section –11.9.3

NOT Identically equal to (!==) (Operator/identity)

Compare two values for non-equality and identical type.

Availability:	ECMAScript edition –3 JavaScript –1.3 JScript –1.0 Internet Explorer –3.02 Netscape –4.06	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`anOperand1`* `!==` *`anOperand2`*
Argument list:	*`anOperand1`*	A value to be compared
	`anOperand2`	Another value of the same type to be compared

The two operands are compared and Boolean `false` is returned if both values are equal, and of the same type, otherwise Boolean `true` is returned.

The associativity is from left to right.

Refer to the Operator Precedence topic for details of execution order.

Warnings:

- This is not available for use server-side with Netscape Enterprise Server 3.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
myObject1 = 100;
myObject2 = "100";
if(myObject1 != myObject2)
{
   document.write("Objects are NOT equal<BR>");
}
else
{
   document.write("Objects are equal<BR>");
}

if(myObject1 !== myObject2)
{
   document.write("Objects are NOT identical<BR>");
}
else
{
   document.write("Objects are identical<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	ASCII, Associativity, Equal to (==), Equality expression, Equality operator, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Identity operator, JellyScript, Less than (<), Less than or equal to (<=), Logical expression, Logical operator, NOT Equal to (!=), Operator Precedence, Relational expression, Relational operator, typeof, Unicode

Cross-references:

ECMA 262 edition 3 –section –11.9.5

Wrox *Instant JavaScript* –page –39

Notation object (Object/DOM)

Notations are decalred in the DTD and are encapsulated in these `Notation` objects.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Inherits from:	`Node` object	
JavaScript syntax:	-	*myNotation* = new Notation*()*
Object properties:	`publicId, systemId`	

See also:	`Doctype` object, `Doctype.notations[]`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`publicId`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly
`systemId`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly

Inheritance chain:

`Node` object

Notation.publicId (Property)

The public identifier (if one was defined) for this notation. The value may be `null` if no identifier was defined.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	myNotation.publicId

Property attributes:

`ReadOnly`.

Notation.systemId (Property)

The system identifier (if one was defined) for this notation. The value may be `null` if no identifier was defined.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNotation*.systemId

Property attributes:

`ReadOnly`.

null (Primitive value)

A built-in primitive value.

Availability:	ECMAScript edition –2 JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`Null primitive`	
JavaScript syntax:	IE	`null`

The `null` value is a primitive value that represents the `null`, empty, or non-existent reference.

This is equivalent to the Java `null` data type when passing values back and forth between JavaScript and Java.

If you don't have a `null` keyword, you may be able to simulate a `null` value like this:

```
var null = (void 0);
```

The `null` and `undefined` values are subtly different. An empty thing is not the same as a non-existent thing. However in a browser it is difficult to distinguish between them.

The `null` value is now provided in some browsers as a built-in keyword, but the `undefined` value is not.

See also:	Cast operator, JavaScript to Java values, LiveConnect, `NaN`, undefined

Cross-references:

ECMA 262 edition 2 –section –4.3.11

ECMA 262 edition 3 –section –4.3.11

O'Reilly *JavaScript Definitive Guide* –page –47

null (Type)

A native built-in type.

Availability:	ECMAScript edition –2
Property/method value type:	`Null primitive`

The type `null` has exactly one value, called `null`.

You can use this value when testing for the undefined state of variables and objects in browsers that do not support an explicit `undefined` value, to test for in logical expressions.

See also:	Cast operator, Special type, Type

Cross-references:

ECMA 262 edition 2 –section –4.3.12

ECMA 262 edition 2 –section –8.2

ECMA 262 edition 3 –section –4.3.12

ECMA 262 edition 3 –section –8.2

Wrox *Instant JavaScript* –page –14

Null literal (Primitive value)

A literal constant whose type is a built-in primitive value.

Availability:	ECMAScript edition –2 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Null primitive`	
JavaScript syntax:	-	`null`

The `null` literal is a value that represents the `null` or undefined state. It only has one value.

Name:	null

The null value is sometimes used in place of other values. For example, in some browser-based interpreters, there is no specific value for the undefined condition. However, you can work around this by testing for null. Strictly speaking they are distinctly different values with different semantic meanings. Even so, the trick works well enough for most practical purposes.

Use null in place of undefined when testing for the existence of entities.

See also:	Literal, Range error, Token, undefined

Cross-references:

ECMA 262 edition 2 –section –7.7.1

ECMA 262 edition 3 –section –7.8.1

Null statement (Definition)

An empty statement consisting of a semicolon on its own.

See also:	Empty statement (;), Semicolon (;)

Number (Primitive value)

A built-in primitive value.

Availability:	ECMAScript edition –2
Property/method value type:	Number primitive

A number value is a member of the type Number and is a direct representation of a number.

It is basically a floating-point number and there is no special integer number class. Integers are simply floating-point values with a zero fractional part.

Numbers can be expressed as integers or floating-point values. They can be expressed in decimal, octal, or hexadecimal notation. They can also be expressed in exponential form.

Typical limits for the number type allow for very large number values. You can find out what the maximum value is by requesting the MAX_VALUE property from the built-in Number object. It will probably give you a value in the region of 1 followed by some 300 or more zeros. The smallest value is some 300 decimal places past the decimal point.

Actually the limits are 1.79e308 down to 5e-324 and both can be positive or negative.

There are special constants for the values `Infinity` and `NaN`. The `Infinity` value can be positive or negative. The `NaN` value represents a quantity that is known to be numeric but is not a valid value for the implementation. It can be tested for with the `isNaN()` function.

See also:	Cast operator, Floating constant, `Infinity`, `isNaN()`, JavaScript to Java values, `Number.MAX_VALUE`, `Number.MIN_VALUE`, Primitive value

Cross-references:

ECMA 262 edition 2 –section –4.3.19

ECMA 262 edition 3 –section –4.3.19

Wrox *Instant JavaScript* –page –14

Number (Type)

A native built-in type.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

The number type defines objects that represent numbers and include the IEEE 754 `NaN` values in the set if the implementation is fully ECMA-compliant.

The number range is huge, representing the double precision 64 bit IEEE 754 set of values.

The IEEE 754 standard defines a large number of values that are considered to be not an actual number, and should be represented as `NaN`. Some implementations of IEEE 754 allow you to tell the difference between an overflow error and a divide by zero error. In JavaScript, all of these values are represented by a single `NaN` value and you cannot distinguish between them. The JavaScript `NaN` value is globally defined as a variable.

The `Number` type includes the values positive and negative Infinity and, internally at least, positive and negative zero are represented as two distinct values.

See also:	Cast operator, Data Type, Floating point, Fundamental data type, Hexadecimal value, IEEE 754, Infinity, Integer, `NaN`, Octal value, Primitive value, `ToBoolean`, `ToInt32`, `ToNumber`, `ToObject`, `ToPrimitive`, `ToUint16`, `ToUint32`, Type, Type conversion, `valueOf()`

Cross-references:

ECMA 262 edition 2 –section –4.3.19

ECMA 262 edition 2 –section –8.5

ECMA 262 edition 3 –section –4.3.20

ECMA 262 edition 3 –section –8.5

O'Reilly *JavaScript Definitive Guide* –page –34

Number formats (.) (Definition)

The period character is used as numeric delimiter character. ECMA describes these formats as Numeric literals.

Availability:	ECMAScript edition –2
See also:	Decimal point (.), Numeric literal

Cross-references:

ECMA 262 edition 2 –section –7.7.3

ECMA 262 edition 3 –section –7.8.3

Number object (Object/core)

An object of the class "`Number`".

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server –2.0 Opera –3.0	
JavaScript syntax:	-	*myNumber* = new Number*()*
	-	*myNumber* = Number
Object properties:	`constructor, prototype`	
Class constants:	`MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY`	
Object methods:	`toExponential(), toFixed(), toLocaleString(), toPrecision(), toSource(), toString(), valueOf()`	

An instance of the class `Number` is created by using the new operator on the `Number()` constructor. The new object adopts the behavior of the built-in `prototype` object through the prototype-inheritance mechanisms.

All properties and methods of the `prototype` are available as if they were part of the instance.

A `number` object is a member of the type `Object` and is an instance of the built-in `Number` object.

`Number` objects are created by cloning the built-in `Number` object. This is done by calling the `Number` constructor with the `new` operator being applied to an existing `Number` object. Thus:

```
myNumber = new Number(1000);
```

A `Number` object can be coerced to a number value and can be used anywhere where a number value would be expected.

Programmers familiar with object-oriented techniques may prefer to use the `Number` object while procedural language programmers may implement the same functionality with a number value instead.

This is an example of the flexibility of JavaScript in its ability to accommodate a variety of users from different backgrounds.

The prototype for the `Number prototype` object is the `Object prototype` object.

You might want to add useful methods to the `Number.prototype` to output numbers in unusual formats. For example you could implement a roman numeral conversion method. Adding that to the prototype would let you output year numbers in classical formats.

Warnings:

- The `Number` object provides a collection of static constant values by way of properties belonging to the integral `Number` object. Because the mathematical mechanisms of any application tend to be provided by the operating system, you should find that between different browsers on any particular platform, the values that these constants yield will be very consistent.
- The ECMA standard lays down strict values for these properties and in general the browser manufacturers try to comply, but there is always the possibility that an implementation may use a non-compliant calculation.
- However, it may not be quite so reliable across platforms. You might enumerate one of these constants as you are authoring and then hard code that value into your script. When that script is executed on another platform, even in the same browser, the internal numeric support may yield a different value.
- You should always refer to the static constants using their symbolic names and not define them yourself, unless you are certain that the script is running on a platform that does not already define the constant value.

See also:	Constant, Limits, `Native` object, `Number.Class`, `Number.prototype`, `Object` object

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`constructor`	1.1 +	1.0 +	3.0 +	3.02 +	-	-	2 +	-
`prototype`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	2.0 +	2 +	DontDelete, DontEnum

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`toExponential()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toFixed()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toLocaleString()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	Warning
`toPrecision()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toSource()`	1.3 +	-	4.06 +	-	-	-	3 +	-
`toString()`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	-	2 +	-
`valueOf()`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	2 +	-

Cross-references:

ECMA 262 edition 2 –section –4.3.20

ECMA 262 edition 2 –section –10.1.5

ECMA 262 edition 2 –section –15.7

ECMA 262 edition 3 –section –4.3.21

ECMA 262 edition 3 –section –10.1.5

ECMA 262 edition 3 –section –15.7

Wrox *Instant JavaScript* –page –33

Number() (Constructor)

A `Number` object constructor.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0	
Property/method value type:	`Number` object	
JavaScript syntax:	-	`new Number()`
	-	`new Number(aValue)`
Argument list:	*`aValue`*	A value to be converted to a `Number` object

The `Number()` constructor is used to manufacture a new instance of the built-in `Number` object converting the input value to a number as it instantiates the new object.

The value of the `Number` object when the constructor is called in a new expression is the same as value yielded by the type conversion function call.

Refer to the `Number()` Function topic for a table of rules for converting non-number values to `Number` objects.

The result is a `Number` object whose value is equivalent to the that of the passed-in argument. If the value is omitted, the `Number` object will assume a value of 0.

Warnings:

- Note that unlike the `Object()` constructor, which can be called without its parentheses, calling the `Number()` constructor without them yields an uninitialized object.

See also:	Constructor function, `constructor` property, `Global` object, new, `Number.prototype`, Object constant, `Object()`

Cross-references:

ECMA 262 edition 2 –section –15.1.3.6

ECMA 262 edition 2 –section –15.7.1

ECMA 262 edition 2 –section –15.7.3.1

ECMA 262 edition 3 –section –15.7.2

Number() (Function)

A Number type convertor.

Availability:	ECMAScript edition –2 JavaScript –1.2 JScript –1.0 Internet Explorer –3.02 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Number()`
	-	`Number(aValue)`
Argument list:	*aValue*	A value to be converted to a number.

When the `Number()` constructor is called as a function, it will perform a type conversion.

The following values are yielded as a result of calling `Number()` as a function:

Value	Result
No value	`0`
Undefined	Returns `NaN`
Null	`0`
Boolean `false`	`0`
Boolean `true`	`1`
Number	No conversion, the input value is returned unchanged.
Non numeric string	`NaN`
Numeric string	The numeric value rounded down if the number of digits exceeds the numeric accuracy specified by `Number.MAX_VALUE`.

Table continued on following page

Value	Result
Object	Internally, a conversion to one of the primitive types happens followed by a conversion from that type to a number. Some objects will return a number that is readily usable; others will return something that cannot be converted and `NaN` will result.

The result is a number value that is equivalent to the value of the passed in argument. If the argument is omitted the value `0` is returned.

Warnings:

- When converting strings to numbers, the number of digits in the numeric string is significant. If it exceeds the accuracy that the numeric storage can cope with, the value needs to be rounded before conversion. This is an area where the implementations are notoriously weak. MSIE apparently does a better job than Netscape. However, both are undergoing revision and its possible that the new versions of each will cope better than the older ones did.

See also:	Cast operator, Constructor function, `constructor` property, Implicit conversion, `Math.ceil()`, `Math.floor()`, `Math.round()`, Number.prototype, `String()`

Cross-references:

ECMA 262 edition 2 –section –15.1.3.6

ECMA 262 edition 2 –section –15.7.1

ECMA 262 edition 3 –section –15.7.1

Wrox *Instant JavaScript* –page –36

Number.Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition –2

This is an internal property that describes the class that a `Number` object instance is a member of. The reserved words suggest that in the future, this property may be externalized.

See also:	Class, `Number` object

Property attributes:

`DontEnum`, `Internal`.

Cross-references:

ECMA 262 edition 2 –section –8.6.2

ECMA 262 edition 2 –section –15.7.2.1

ECMA 262 edition 3 –section –8.6.2

Number.constructor (Property)

A reference to a `constructor` object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0	
Property/method value type:	`Number` object	
JavaScript syntax:	-	*myNumber*`.constructor`

The initial value of the `Number` constructor is the built in `Number` object.

You can use this as one way of creating number objects although it is more popular to use the new `Number()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective, and there are some occasions when you might wish for a constructor that is not available in MSIE.

See also:	`Number.MAX_VALUE`, `Number.MIN_VALUE`, `Number.NaN`, `Number.NEGATIVE_INFINITY`, `Number.POSITIVE_INFINITY`, `Number.prototype`

Cross-references:

ECMA 262 edition 2 –section –15.7.2

ECMA 262 edition 3 –section –15.7.2

Number.MAX_VALUE (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Number.MAX_VALUE`

This is a constant value representing the largest realizable positive finite value of the `Number` type.

The value is approximately 1.7976931348623157e+308.

Warnings:

- The word approximately is used when describing the result, because the mathematical accuracy of JavaScript implementations leaves something to be desired and there are some strange artifacts in some of the calculations.

See also:	Arithmetic constant, Floating point constant, Number, `Number.constructor`, Special number values

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.7.3.2

ECMA 262 edition 3 –section –15.7.3.2

O'Reilly *JavaScript Definitive Guide* –page –37

Number.MIN_VALUE (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Number.MIN_VALUE`

This is a constant value representing the smallest realizable non-zero positive value of the number type.

The resulting value is approximately 5e-324

Warnings:

- Although the `MAX_VALUE` is generally correct across implementations there are variations in the value of the `MIN_VALUE` constant.

See also:	Arithmetic constant, Floating point constant, Number, `Number.constructor`, Special number values

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.7.3.3

ECMA 262 edition 3 –section –15.7.3.3

O'Reilly *JavaScript Definitive Guide* –page –37

Number.NaN (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Number.NaN`

This is a value representing invalid numeric values. It should be identical to the `NaN` value provided by the `Global` object in an ECMA-compliant implementation. Refer to the coverage of the `NaN` topic for full details.

However, it is generally considered unreliable to compare against `NaN` values with a simple equality test. To reliably test whether a numeric value is `NaN` or a good numeric value, use the `isNaN()` function and select an appropriate action according to its result.

See also:	Arithmetic constant, `NaN`, `Number.constructor`, Special number values

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.7.3.4

ECMA 262 edition 3 –section –15.7.3.4

O'Reilly *JavaScript Definitive Guide* –page –37

Number.NEGATIVE_INFINITY (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Number.NEGATIVE_INFINITY`

This is a constant representing the value negative infinity. It should be identical to a negative sign placed in front of the Infinity value provided as a property of the `Global` object in an ECMA-compliant implementation. Refer to the Infinity topic for further discussion.

Warnings:

- Netscape 2.02 does not understand what `Number.NEGATIVE_INFINITY` is.

See also:	Arithmetic constant, Infinity, `Number.constructor`, Special number values

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.7.3.5

ECMA 262 edition 3 –section –15.7.3.5

O'Reilly *JavaScript Definitive Guide* –page –37

Wrox *Instant JavaScript* –page –15

Number.POSITIVE_INFINITY (Constant/static)

A mathematical constant value.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`Number.POSITIVE_INFINITY`

This is a constant value representing positive infinity. It should be identical to the Infinity value provided as a property of the `Global` object in an ECMA-compliant implementation. Refer to the Infinity topic for further discussion.

Warnings:

- Netscape 2.02 does not understand what `Number.POSITIVE_INFINITY` is.

See also:	Arithmetic constant, Infinity, `Number.constructor`, Special number values

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.7.3.6

ECMA 262 edition 3 –section –15.7.3.6

O'Reilly *JavaScript Definitive Guide* –page –37

Wrox *Instant JavaScript* –page –15

Number.prototype (Property)

The prototype for the `Number` object that can be used to extend the interface for all `Number` objects.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number` object	
JavaScript syntax:	-	`Number.prototype`
	-	*`myNumber`*`.constructor.prototype`

The `prototype` property refers to the built-in `Number` `prototype` object.

The example demonstrates how to provide extensions to all instances of this class by adding a function to the `prototype` object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that extends the capabilities of the Number object
function pythag(aValue1, aValue2)
{
   return Math.sqrt((aValue1*aValue1) + (aValue2*aValue2));
}

// Register the new function
Number.prototype.pythag = pythag;

// Create a number object and test the Number.pythag() method
myNumber = new Number();
document.write(myNumber.pythag(3, 4));
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Number` object, `Number()`, `Number()`, `Number.constructor`, `Number.toString()`, `Number.valueOf()`, `prototype` property

Property attributes:

`DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.2.3.1

ECMA 262 edition 2 –section –15.7.4

ECMA 262 edition 3 –section –15.7.3.1

Number.toExponential() (Method)

Converts the number to an exponential format representation.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNumber*`.toExponential(`*aNumber*`)`
Argument list:	*aNumber*	The number of digits after the decimal point

This method is useful for formatting number values. This is especially helpful when presenting tables of scientific data, which may require values of a wide range of magnitudes to be presented in a similar way.

The argument value indicates the precision or decimal places of accuracy to the right of the decimal point character. If the argument is undefined, then as many digits as are necessary to completely distinguish the value are presented (up to the mathematical accuracy of the implementation).

See also:	`Number.toFixed(),Number.toPrecision(), Number.toString()`

Cross-references:

ECMA 262 edition 3 –section –15.7.4.6

Number.toFixed() (Method)

Converts the number to a fixed format representation.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNumber*`.toFixed(`*aNumber*`)`
Argument list:	*aNumber*	The number of digits after the decimal point

This method is useful for truncate formatting number values. This is especially helpful when presenting tables of financial data that needs to be justified and padded to the same number of digits after the decimal point.

The argument value indicates the precision or decimal places of accuracy to the right of the decimal point character. If the argument is undefined, then zero is assumed and the value will be presented as an integer.

This method may be useful for performing truncations from floating point to integer value. The output of this method may also be more precise when a large number of integer digits are required to present the number. According to the ECMA standard, the alternative `toString()` method loses some accuracy for numbers having 19 digits for example.

See also:	`Number.toExponential(),Number.toPrecision(), Number.toString()`

Cross-references:

ECMA 262 edition 3 –section –15.7.4.5

Number.toLocaleString() (Method)

Converts a number to a string taking locale-specific settings into account.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNumber*`.toLocaleString()`

Locale conventions may include special formatting of numbers such as thousands separators and decimal point symbols. The number will be converted according to the rules defined by the implementation's locale setting.

Warnings:

- ECMA warns that the first argument of this method is reserved for future use. Beware if your implementation makes use of an argument in this method as it may be non-compliant with a later version of the ECMA standard.

Cross-references:

ECMA 262 edition 3 –section –15.7.4.3

Number.toPrecision() (Method)

Convert a number to a string automatically selecting fixed or exponential notation.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNumber*`.toPrecision(`*aNumber*`)`
Argument list:	*aNumber*	The number of digits after the decimal point

This method will convert a number to a string and will select either a fixed or exponential notation according to the magnitude of the value being converted.

This would be useful where you have an arbitrary collection of values and don't want them presented in a ragged looking column.

See also:	`Number.toExponential(),Number.toFixed()`

Cross-references:

ECMA 262 edition 3 –section –15.7.4.7

Number.toSource() (Method)

Output a number formatted as a Number literal contained in a string.

Availability:	ECMAScript edition –3 JavaScript –1.3 Netscape –4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNumber*`.toSource()`

This is an alternative way to deliver a string version of a number value. In this case, it is formatted as a Number literal and can then be used in an `eval()` function to assign another number.

If you run the example below, it should yield this as a result:

```
(new Number(1000))
```

The result of calling this method is a string version of the number formatted as a Number literal.

Example code:

```
// Create a number and then examine its source
myNumber = new Number(1000);
document.write(myNumber.toSource());
```

Number.toString() (Method)

Return a string primitive version of an object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myNumber*`.toString(`*aRadix*`)`
Argument list:	*aRadix*	A radix to base the string conversion of the value on

The result of this method is a String primitive representation of the numeric value of the receiving object, rendered according to the passed-in radix value.

A radix is the number of discrete values in the counting sequence before a carry over digit is generated. Thus, the radix of decimal numbers is 10. With this mechanism, you can generate values to any radix. The radix is also called the base of the number system.

The radix value is in the range 2 to 36 and allows the numeric value to be reproduced in a variety of number bases.

- A missing radix is assumed to be base 10.
- Octal numbers use a radix value of 8.
- Hexadecimal numbers use a radix value of 16.
- Binary numbers use a radix value of 2.

For an example, refer to the Decimal value topic where this is used to generate a lookup table.

Before the radix conversion was available, this method simply converted to a string, which most commentators considered was unnecessary since JavaScript did this naturally without any need to add special scripting support. However, now that we can convert numeric values from one base to another, this function becomes far more useful.

See also:	Cast operator, Decimal value, Hexadecimal value, `Number.prototype`, `Number.toExponential()`, `Number.toFixed()`, Octal value, `toString()`

Cross-references:

ECMA 262 edition 2 –section –15.7.4.2

ECMA 262 edition 3 –section –15.7.4.2

Number.valueOf() (Method)

Return the primitive numeric value of the object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myNumber*`.valueOf()`

A `Number` object is converted to a simple Number primitive. You probably won't need to do this very often yourself because JavaScript is smart enough to convert Number primitives to `Number` objects and vice versa whenever it needs to. This is a whole lot better than having to cast data types to get expressions to work, as you would have to in other programming languages. This help makes JavaScript very accessible to non-programmers.

See also:	Cast operator, `Number.prototype`, `valueOf()`

Cross-references:

ECMA 262 edition 2 –section –15.7.4.3

ECMA 262 edition 3 –section –15.7.4.4

Numeric literal (Primitive value)

A literal constant whose type is a built-in primitive value.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

Numeric literals are constant numeric values expressed in Decimal, Hexadecimal, or Octal notation.

Numeric values can be integer or floating-point.

Floating-point values can be specified with exponential notation.

Hexadecimal values must always be integers, thus:

- `0xFF`
- `0XABCD`

Octal values must always be integers, must start with a zero and contain only the characters 0-7.

The standard does not mandate any particular rounding technique but recommends the use of IEEE 754 standard numeric computation. This standard has been in existence for some time now and is likely to be the foundation numeric computation standard in most platforms.

See also:	Floating constant, Implicit conversion, Literal, Number formats (.)

Cross-references:

ECMA 262 edition 2 –section –7.7.3

ECMA 262 edition 3 –section –7.8.3

Numerical limits (Definition)

Limiting conditions for arithmetic values.

Refer to:

Limits

Obfuscation (Advice)

Needless complexity in the arrangement of tokens in a line of executable script.

There are competitions on the Internet to write the most egregious and highly functional code fragment in the fewest lines of code. This is particularly easy to do in C language. Certain operators lend themselves to the construction of extremely terse code, which, although functionally very clever, is also very hard to understand when diagnosing faults or carrying out maintenance.

This entry is not to recommend against the use of such operators, but to urge a word of caution on the basis that modern language parsing and execution engines may be smart enough to highly optimize the code, making any performance gains negligible anyway.

These days, it is quite difficult to yield any noticeable performance gains by epitomizing the code at the source level with compiled systems. There may still be some gains to be won with interpreted code. A badly designed algorithm may continue to perform badly in an interpreter where there is a possibility it might get corrected in an optimizing compiler.

The operators to be particularly cautious about are the assignment operators and the prefix/postfix operators. The ternary conditional operator is also hard to read in source and may offer little advantage over an `if(...) ...else` construct. These are equivalent but intent is much more obvious withthe `if(...) ... else` form:

```
// Conditional ternary operator

myResult = (mySwitch) ? "TRUE VALUE" : "FALSE VALUE" ;

// Conditional block

if(mySwitch)

{ myResult = "TRUE VALUE";}

else

{ myResult = "FALSE VALUE";}
```

Where this causes particular problems is in the maintenance phase where you might perhaps be adding another line of code. In cutting and pasting an existing line, it can be easy to overlook an operator and accidentally increment something twice or assign a value inadvertently.

Special care is necessary with iterators and conditional execution blocks. A particularly nasty habit is to have a condition that when `true`, executes one statement. This is frequently written into the source text without any enclosing braces. Those braces are important because they group the block of code into a single syntactical unit. When you later try to unpick someone else's script, the indentation may fool you into seeing several lines that appear to be conditionally executed when in fact only one is. The same applies to iterators as well. It is highly recommended that you put in the braces where they are required for multiple line conditional code and iterator blocks even when there is only one line of code being executed. This safeguards against errors when more lines are added to the conditional or iterated code block later on.

This is not recommended practice:

```
if(aCondition)

someCode;

while(aCondition)

someCode;

for( ... )

someCode;
```

This is slightly better but requires more effort when adding lines to the code block:

```
if(aCondition) someCode;

while(aCondition) someCode;

for( ... ) someCode;
```

This is less dangerous than having no braces but makes the line long and twice as hard to scan visually:

```
if(Condition) { someCode }
```

This is fashionable, but the braces are hard to balance visually:

```
if(aCondition) {

someCode

}
```

This is good because it thinks ahead to the possibility of maintenance and tends towards fewer editing errors:

```
if(aCondition)

{
someCode

}

while(aCondition)

{

someCode

}

for( ... )

{

someCode

}
```

If there is any downside to this it is that every balanced pair of braces will create a new execution context. This may slow performance, but on the other hand it can allow locally scoped variables to be created and destroyed at a level that is more granular than a function body.

There is also scope for a religious debate on indentation. Three space characters works great (for me). I don't like tabs because if you move the source code to another editor, the indentation can go awry. Space characters for indentation ensure that the source code looks the same in any monospaced editing window and probably looks OK in a word processor too.

See also:	Flow control, `if( ... ) ..., while( ... ) ...`

Object (Definition)

There is a distinct difference between an object and an Object.

We refer to the built-in `Object` class with a capitalised name. When referring generically to objects of other classes, the word object is all lower case. Therefore we can have an `Object` object and a `String` object. Native objects are built-in, host objects are also built in but created outside of the JavaScript core functionality. User-defined objects are not covered here.

Here is a list of object classes with a note about what sort of object they are and when how they are managed:

Class	Category	Description
`Array`	Native	A collection of objects in a sequence
`Boolean`	Native	A logical value container
`Date`	Native	A date value container
`Function`	Native	A function code container
`Global`	Built-in	A container for global properties, methods, and functions
`Image`	Hosted	Web browser image wrapper
`Math`	Built-in	A container for math functions
`Number`	Native	A numeric value
`Object`	Native	A generic object
`String`	Native	A sequence of characters

Because you might refer to documents in many ways, possibly by means of object properties or as a property belonging to another window, it is not safe to assume that the document property belonging to the `Global` object the script is attached to is always the document object you are trying to access. Because of this, the object references in the syntax examples assume the object is being referred to via a variable called *`myDocument`* or *`myObject`* etc. For example, the value *`myDocument`* is shown being assigned as a variable from the many alternative sources from which you can obtain a document object reference.

Object (Type)

A native built-in type.

Availability:	ECMAScript edition –2

An Object is an unordered collection of properties. Each property consists of a name, a value and a set of attributes.

See also:	Alias, Data Type, Definition, Internal Method, Internal Property, Object object, Property, Property attribute, Type

Cross-references:

ECMA 262 edition 2 –section 8.6

ECMA 262 edition 3 –section 8.6

Wrox *Instant JavaScript* –page 28

Object constant (Definition)

Constant objects.

There are some constant objects defined by the standard. A host implementation may provide a few more.

Here is a summary of the basic set of constant objects:

- `Array` constructor
- `Boolean` constructor
- `Date` constructor
- `Function` constructor
- `Global` object
- `Math` object
- `Number` constructor
- `Object` constructor
- `RegExp` constructor

See also:	`Array()`, `Boolean()`, `Closure()`, Constant expression, `Date()`, `Function()`, `Global` object, `Math` object, `Number()`, Object literal, Object`()`

Cross-references:

Wrox *Instant JavaScript* –page 28

Object inspector (Useful tip)

A debugging tool for inspecting object properties and classes.

Here is a small debugging utility that breaks an object down and displays some of its properties.

Example code:

```
<!-- An example for use on Netscape only -->
<HTML>
<HEAD>
<SCRIPT>
function object_dump(anObject)
{
   document.write("<HR>");
   document.write("<TABLE BORDER=1>");
   table_row("<B>Item</B>", "<B>Value</B>");

   if(typeof anObject != "object")
   {
```

```
      table_row("Value", anObject);
      table_row("Typeof", typeof(anObject));
   }
   else
   {
      table_row("Value", anObject.valueOf());
      table_row("String equiv", anObject.toString());
      table_row("Typeof", typeof(anObject));
      table_row("Class", anObject.constructor.name);
      table_row("Prototype", anObject.prototype);
   }

   document.write("</TABLE>");
   document.write("<HR>");
}

function table_row(anItem, aName)
{
   document.write("<TR>");
   document.write("<TD>");
   document.write(anItem);
   document.write("</TD>");
   document.write("<TD>");
   document.write(aName);
   document.write("</TD>");
   document.write("</TR>");
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
function nest(abc)
{
   test("aa", bb, 100);
}
function test(aa, bb, cc, dd)
{
   object_dump(aa);
   object_dump(bb);
   object_dump(cc);
   object_dump(arguments.caller.callee.name);
   object_dump(arguments.callee.constructor.name);
}
var bb;
nest("aa", bb, 100);
object_dump(document);
</SCRIPT>
</BODY>
</HTML>
```

See also:	Arguments object, Debugging –client side, typeof

Object literal (Definition)

An object initialiser that creates the object as well.

Availability:	ECMAScript edition –3
Property/method value type:	`Object` object

JavaScript version 1.2 introduces Object literals.

The object is created and returned by the expression. This would normally be assigned to a variable, which effectively names the object. It isn't the object class but it can be copied. Its class is still "`Object`".

Object literals can be nested so that the properties of the topmost object can in fact be object literals themselves.

The values assigned to the properties as the object is created can be derived by evaluating a JavaScript expression.

You can add as many properties as you care to but you must be careful to keep the nesting properly balanced.

The result is an object with the properties containing the values described in the literal expression.

Example code:

```
// Create a simple object literal
var simple = { prop:100 };
// Create a nested object literal
var nested = { reference: { prop:100 } };
// Create a nested object literal with expression derived value
var evaluated = { reference: { prop:(Math.random()*100) } };
```

See also:	Object constant, `Object.constructor`

Cross-references:

ECMA 262 edition 3 –section –11.1.5

O'Reilly *JavaScript Definitive Guide* –page –45

Object model (Definition)

There are several different object models that are realized in JavaScript implementations.

In the Netscape and MSIE web browsers, the object models are provided as representations of the document, the browser, event capturing mechanisms, and the style sheet. In addition, some implementations model the environs, the operating system, and the file system. Each of these object models interacts with the others and is a way of representing the tangible real-world objects.

Although these are generally arranged in a tree-like structure, there are many short cut references that mean you can refer to the same object in a variety of ways. For example the Netscape `JavaPackage` object can be referred to with the following properties in a Netscape browser:

- `Netscape`
- `Packages.netscape`
- `window.Packages.netscape`

Each one refers to an identical object but from the script writer's point of view, some time can be saved by using the short cuts. Scripts also appear simpler to read.

However, the downside is that the object model hierarchy becomes confusing unless you know about the short-cuts. These shortcuts provided for 'so-called' convenience may in fact be exactly the opposite if they are only available on one platform. Using them immediately renders your script non-portable.

Object object (Object/core)

An object of the class "`Object`".

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Netscape Enterprise Server 2.0 Opera 3.0	
JavaScript syntax:	-	*myObject* = new Object()
	-	*myObject* = Object
Object properties:	`__parent__, __proto__, constructor, name, prototype`	
Object methods:	`assign(), eval(), hasOwnProperty(), isPrototypeOf(), propertyIsEnumerable(), toLocaleString(), toSource(), toString(), unwatch(), valueOf(), watch()`	

An instance of the class "`Object`" is created by using the new operator on the `Object()` constructor. The new object adopts the behavior of the built-in prototype object through the prototype-inheritance mechanisms.

All properties and methods of the prototype are available as if they were part of the instance.

An Object is a member of the type `Object`. It is an unordered collection of properties, each of which may contain a primitive value, another object, or a function.

The constructor is invoked by the `new` operator or by calling it as a Constructor function. For example:

```
new String("Some Text");
```

This will create a new object of the `String` type. You can invoke the constructor without the new operator but the consequences will depend on the constructor as to what it will yield as a result. In the case of the `String` data type, the constructor could be invoked like this:

```
String("Some Text");
```

However, you would get a primitive string as a result from this and not a `String` object. JavaScript is somewhat forgiving and you may not notice this happening until later on when it becomes important that you have a `String` object and not a simple string.

Because this object is the topmost parent object in the prototype inheritance hierarchy, all other object classes inherit its methods and properties. However, in some cases they will get overridden or nulled out.

DOM level 2 adds the following properties:

`contentDocument`

Although JavaScript is object-based, it does not support true object-oriented classes such as the ones you find in C++, Smalltalk, Java or Objective C. Instead, it provides Constructor mechanisms, which create objects by allocating space for them in memory and assigning initial values to their properties.

All functions, including constructors are themselves objects; however, not all objects are constructors. Each constructor has a Prototype property that is used to facilitate inheritance based on the prototype. It also provides for shared properties, which is similar to but not the same as the Class properties that you find in true object-oriented languages.

Externally, the objects in JavaScript exhibit most of the attributes of a class based object oriented system and some commentators argue that this qualifies JavaScript as being a genuine object oriented system. However I think the following points declassify it as a truly object oriented system, meaning that it is an "object like" system:

- Global variables and the scope chain mechanism
- Prototype based inheritance
- Creation of multiple objects and calling them within a single script
- Object data is not truly private

It's a close enough call that JavaScript 2.0 may well move it into the class-based object-oriented category at which time the prototype inheritance would be replaced with super-class/sub-class mechanisms and the arguments become null and void.

Warnings:

- Be very careful not to confuse this generic top-level core object with the object that MSIE instantiates to represent an `<OBJECT>` tag. MSIE creates `OBJECT` objects for that purpose but also supports `Object` objects. For this reason, it may be the case that interpreters cannot become case-insensitive when matching class names. If they did, then it would be impossible to distinguish between Object and OBJECT class names.

See also:	Aggregate type, `Array` object, `Boolean` object, `Date` object, delete, `Function` object, `Math` object, Native object, `Number` object, Object, `OBJECT` object, `Object`(), Object(), `Object.Class`, `Object.prototype`, `String` object, `userDefined` object

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`__parent__`	1.2 +	-	4.0 +	-	-	-	-	-
`__proto__`	1.2 +	-	4.0 +	-	-	-	-	-
`constructor`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	2 +	-
`name`	-	5.5 +	-	5.5 +	-	-	-	-
`prototype`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	2 +	Warning, ReadOnly, DontDelete, DontEnum

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`assign()`	1.1 +	-	3.0 +	-	-	-	-	Warning, Deprecated
`eval()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	-	Warning, Deprecated
`hasOwn Property()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`isPrototype Of()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`propertyIs Enumerable()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`toLocale String()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	Warning
`toSource()`	1.3 +	3.0 +	4.06 +	4.0 +	-	-	-	Warning
`toString()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	2 +	-
`unwatch()`	1.2 +	-	4.0 +	-	-	3.0 +	-	Warning
`valueOf()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	-
`watch()`	1.2 +	-	4.0 +	-	-	3.0 +	-	Warning

Cross-references:

ECMA 262 edition 2 –section 4.2.1

ECMA 262 edition 2 –section 4.3.3

ECMA 262 edition 2 –section 10.1.5

ECMA 262 edition 2 –section 15.2

ECMA 262 edition 3 –section 4.2.1

ECMA 262 edition 3 –section 4.3.3

ECMA 262 edition 3 –section 10.1.5

ECMA 262 edition 3 –section 15.2

O'Reilly *JavaScript Definitive Guide* –page 44

Wrox *Instant JavaScript* –page 28

Object() (Constructor)

An `Object` object constructor.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Netscape Enterprise Server 2.0 Opera 3.0	
Property/method value type:	`Object` object	
JavaScript syntax:	-	`new Object`
	-	`new Object()`
	-	`new Object(aValue)`
Argument list:	*aValue*	An initial value for the new object

When `Object` is called as part of a new expression, it is a constructor that may create an object instance.

There are limitations what is sensible for the `new` operator to be able to do with the `Object` constructor. Since the `Object` is considered to be the highest ancestor of all objects in the prototype inheritance chain, you cannot logically have more than one `Object` object. Passing other native objects to the `Object` Constructor implies a type casting from their native object type to the `Object` type. That's not logical either. The main use of the `Object` Constructor then is to manufacturer object instantiations from non-object data types.

The table summarizes the resulting values from using the `Object()` constructor with the `new` operator.

Value	Result
No argument	Creates a new empty object
null	Creates a new empty object
undefined	Creates a new empty object
Boolean	Create a new boolean object whose default value is the input value
Number	Create a new number object whose default value is the input value
String	Create a new string object whose default value is the input value
Native object	Return the native object itself
Host object	Host implementation dependant behavior. Objects are cloned if necessary but some may not be

Unless you assign the result of the new operation, an object will simply consume memory. You need to store a reference to it at the time it is instantiated. You can do this by assigning it to a variable or a property of another object, passing it in to a function and making sure it gets retained in there, or storing it as an element in an array.

Warnings:

- You can refer to objects without the parentheses but then you are not referring to a constructor function but to the object itself. The behavior varies between browsers and depends on the kind of object being instantiated and probably depends on whether it is a wrapper for a primitive data type or a more complex aggregated type.
- These all appear to create the same kind of object in both MSIE and Netscape

```
myObject = Object();

myNewObject = new Object();

myOtherObject = new Object;
```

  These do not:

```
myBoolean = Boolean();

myNewBoolean = new Boolean();

myOtherBoolean = new Boolean;
```

- In the case of the Boolean, Number, and String object types, only the first form will initialize the new object with a value. Placing the result of these examples in `document.write()` statements illustrates the behavior. You may want to examine the objects returned in more detail by developing an object inspector script.

See also:	`Boolean()`, Constructor function, constructor property, Garbage collection, `Global` object, Memory leak, `new`, `Number()`, Object constant, `Object` object, `Object.prototype`, Reference counting, `String()`

Cross-references:

ECMA 262 edition 2 –section 15.1.1

ECMA 262 edition 2 –section 15.1.3.1

ECMA 262 edition 2 –section 15.2.2.2

ECMA 262 edition 3 –section 15.2.2

Object() (Function)

An `Object` object constructor.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0	
Property/method value type:	An object of a type that depends on the passed in argument	
JavaScript syntax:	-	`Object()`
	-	`Object(aValue)`
Argument list:	*aValue*	A value to be stored in the new object.

The Object Constructor can be called as a function. When this happens, the value passed in undergoes a type conversion.

In an ECMA-compliant implementation, the Object constructor function uses the `ToObject` conversion. However it handles input values `undefined` and `null` as special cases and creates a new object as if the constructor had been used with the `new` operator.

The table summarizes the results based on the input value data types.

Value	Result
No argument	Creates a new empty object as if new `Object()` had been called.
`null`	Creates a new empty object as if new `Object(null)` had been called.
`undefined`	Creates a new empty object as if new `Object(undefined)` had been called.
`Boolean`	Create a new boolean object whose default value is the input value.
`Number`	Create a new number object whose default value is the input value.
`String`	Create a new string object whose default value is the input value.
`Object`	No conversion, the input value is returned unchanged.

See also:	Cast operator, Constructor function, constructor property, Implicit conversion, `Object` object

Cross-references:

ECMA 262 edition 2 –section 15.1.1

ECMA 262 edition 2 –section 15.1.3.1

ECMA 262 edition 2 –section 15.2.2.2

ECMA 262 edition 3 –section 15.2

Object.__parent__ (Property)

A special property in which to access the scope chain during function execution.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`ScopeChain object`	
JavaScript syntax:	N	*myObject*`.__parent__`

See also:	Lexical scoping, `__parent__`

Object.__proto__ (Property)

A special property in which to access the prototype inheritance chain during construction.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Function` object	
JavaScript syntax:	N	*myObject*`.__proto__`

See also:	Lexical scoping, Prototype Based Inheritance, Prototype chain, `__proto__`

Object.assign() (Method)

A deprecated mechanism for intercepting messages sent to objects.

Availability:	JavaScript –1.1 Netscape –3.0 Deprecated	
JavaScript syntax:	N	*myObject*`.assign()`

Warnings:

- This method is deprecated in favor of the `Object.watch()` and `Object.unwatch()` methods.

See also:	`Object.unwatch(), Object.watch()`

Object.Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition –2

This is an internal property that describes the class that an `Object` object instance is a member of. The reserved words suggest that in the future, this property may be externalized.

See also:	Class, `Object` object

Property attributes:

`DontEnum, Internal.`

Cross-references:

ECMA 262 edition 2 –section –8.6.2

ECMA 262 edition 2 –section –15.2.2.1

ECMA 262 edition 3 –section –8.6.2

Object.constructor (Property)

A reference to a constructor object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`Object` object	
JavaScript syntax:	-	*myObject*`.constructor`

The initial value of the `Object.prototype.constructor` is the built-in Object constructor.

You can use this as one way of creating objects although it is more popular to use the new `Object()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	Object literal, `Object.prototype`

Cross-references:

ECMA 262 edition 2 –section –15.2.4.1

ECMA 262 edition 3 –section –15.2.2

Object.eval() (Method)

Evaluate the JavaScript source text passed in a string argument.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Netscape Enterprise Server version –2.0 Opera browser –3.0 Deprecated	
Property/method value type:	Function result	
JavaScript syntax:	-	*myObject*.eval()

Warnings:

- This is now deprecated and may even be unavailable in recent versions of Netscape and JScript. You should use the `eval()` function available from the `global` object.

See also:	`eval()`

Object.hasOwnProperty() (Method)

A method that can be used to test whether a property exists and belongs to the receiving object.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.hasOwnProperty(*aName*)
Argument list:	*aName*	The name of a property to test for

For this method to yield a Boolean `true` value, the property named in the argument must exist and must belong to the receiving object. If the property is inherited from a prototype or earlier ancestor then this method returns `false`.

This method would be useful if it could test for the existence of a property in the inheritance chain. There is an internal `HasProperty()` method but the ECMA standard indicate that it is not exposed to the script interface.

See also:	`HasProperty()`

Cross-references:

ECMA 262 edition 3 –section –15.2.4.5

Object.isPrototypeOf() (Method)

A test for the relationship between two objects to ascertain direct parentage.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.isPrototypeOf(*anObject*)
Argument list:	*anObject*	The object whose prototype is to be tested

The receiving object is tested for identity against the object referred to by the prototype property of the object passed as an argument. If the object in the argument is a direct child object through the prototype chain, then this method returns a `true` value.

See also:	`HasInstance()`

Cross-references:

ECMA 262 edition 3 –section –15.2.4.6

Object.name (Property)

This corresponds to the NAME attribute of the tag that creates the object.

Availability:	JScript –5.5 Internet Explorer –5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myObject*.name

Objects are identified either by the `NAME="..."` HTML tag attribute or by the `ID="..."` HTML tag attribute.

Netscape shows a marginal preference for the `name` property while MSIE seems slightly better disposed towards the `ID` property. However in many cases, both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

See also:	`NAME="..."`

Object.propertyIsEnumerable() (Method)

A test for whether a property has the don't enumerate flag set or not.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*`.propertyIsEnumerable(`*aName*`)`
Argument list:	*aName*	The name of the object property to test

If the receiving object has a member property of the name that is passed in the argument, and if the `DontEnum` attribute of that property is `false`, then this method returns the Boolean `true` value.

Cross-references:

ECMA 262 edition 3 –section 15.2.4.7

Object.prototype (Property)

The prototype for the `Object` object, which can be used to extend the interface for all `Object` objects.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`Object` object	
JavaScript syntax:	-	`Object.prototype`
	-	*myObject*`.constructor.prototype`

The initial value of the prototype property of an `Object` object is the built in `Object` prototype object.

`Object` objects inherit the following properties from the `Object.prototype`:

- `Object.constructor`
- `Object.prototype`

`Object` objects inherit the following methods from their prototype:

- `Object.toString()`
- `Object.valueOf()`

The prototype property for the `Object` prototype object is `null`.

The example demonstrates how to provide extensions to all instances of this class by adding a function to the prototype object.

Warnings:

- This is not supported on the WebTV platform.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that extends the output capabilities of Object objects
function init()
{
   this.width  = 100;
   this.height = 200;
   this.depth  = 300;
   return "";
}
// Register the new function
Object.prototype.init = init;
// Create a new object
myObject = new Object();
myObject.init();
document.write(myObject.width  + "<BR>");
document.write(myObject.height + "<BR>");
document.write(myObject.depth  + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also: `Arguments` object, `Function.arguments[]`, JellyScript, `Object` object, `Object()`, `Object.constructor`, `Object.toString()`, `Object.valueOf()`, `prototype` property

Property attributes:

`ReadOnly, DontDelete, DontEnum.`

Cross-references:

ECMA 262 edition 2 –section 15.2.3.1

ECMA 262 edition 2 –section 15.2.4

ECMA 262 edition 3 –section 15.2.3.1

Object.toLocaleString() (Method)

Returns a string primitive version of the object taking the present locale into account during the translation.

Availability:	ECMAScript edition –3 JavaScript –1.5 JScript –5.5 Internet Explorer –5.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myObject`*`.toLocaleString()`

The locale context supplies some special conversion rules for strings. Depending on the locale, this might include special characters or a means of using double byte characters. It may also affect the direction of the text, for certain Asian locales for example.

Warnings:

- The ECMA standard reserves the first argument of this method for future use. It does not specify what that is but warns against implementations extending the syntax to include its use.

Cross-references:

ECMA 262 edition 3 –section –15.2.4.3

Object.toSource() (Method)

Output a string describing the object contents.

Availability:	JavaScript –1.3 JScript –3.0 Internet Explorer –4.0 Netscape –4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myObject`*`.toSource()`

This is an alternative way to deliver a string version of an object's internal values. In this case, it is formatted as an Object literal and can then be used in an `eval()` function to assign another object.

The exact format of what you see depends on the object being examined.

The result of calling this method is string version of the object formatted as an Object literal.

Warnings:

- Note that this is not available in the MSIE browser but can be useful when constructing an Object inspector for use in Netscape.

Object.toString() (Method)

Return a string primitive version of an object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myObject*`.toString()`

When the `toString()` method of an `Object.prototype` is invoked, the class name of the object is returned as a string.

The result of calling this method will be the string:

```
[object "Object"]
```

See also:	Cast operator, `Object.prototype, toString()`

Cross-references:

ECMA 262 edition 2 –section –15.2.4.2

ECMA 262 edition 3 –section –15.2.4.2

Object.unwatch() (Method)

A method to disable a watch that was set up on a property change.

Availability:	JavaScript –1.2 Netscape –4.0 Netscape Enterprise Server –3.0	
JavaScript syntax:	N	*myObject*.unwatch()

This is inherited by most object classes in Netscape.

Warnings:

- Because of the scoping rules, you cannot actually examine the value of the variable that has changed since the event handler is running in a completely different scope chain.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
function watchFunct()
{
  alert("Variable changed");
}
watch("myVariable", watchFunct)
myVariable = 100;
myVariable = 200;
unwatch("myVariable")
myVariable = 300;
</SCRIPT>
</BODY>
</HTML>
```

See also:	Object.assign(), Object.watch()

Object.valueOf() (Method)

The primitive numeric value of the object.

Availability:	ECMAScript edition –2 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	Depends on the object value	
JavaScript syntax:	-	*myObject*.valueOf()

As a general rule, the `valueOf()` method for an object simply returns the this property of the object itself. However, the object may be a wrapper for a host object some kind. It may therefore have been created by invoking the Object constructor. In that case, the host object should be returned in an ECMA-compliant implementation.

Implementations may choose to return the this property of an object or some other value if they choose.

The result of this method will be implementation-and object-dependant. The native core objects are well defined and will return predictable value types. Generally these will be defined by ECMA or W3C standards. It is up to the hosting environment to provide the `valueOf()` interface to its own suite of objects.

See also:	Cast operator, `Object.prototype, valueOf()`

Cross-references:

ECMA 262 edition 2 –section –15.2.2.1

ECMA 262 edition 2 –section –15.2.4.3

ECMA 262 edition 3 –section –15.2.4.4

Object.watch() (Method)

A means of establishing a call back when a property value changes.

Availability:	JavaScript –1.2 Netscape –4.0 Netscape Enterprise Server version –3.0	
JavaScript syntax:	N	*myObject*`.watch()`

This is inherited by most object classes in Netscape.

Warnings:

- Because of the scoping rules, you cannot actually examine the value of the variable that has changed since the event handler is running in a completely different scope chain.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
document.watch("myVariable", alert("Watch point triggered"))
document.myVariable = 100;
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Object.assign(), Object.unwatch(), unwatch(), watch()`

Object property delimiter (.) (Delimiter)

A token to delimit object properties from their object.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.0 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera browser –3.0	
JavaScript syntax:	-	`myObject.aProperty`
	-	`myObject.aProperty.aProperty`
Argument list:	`aProperty`	The identifier name of property to be accessed

The dot delimits properties and objects. It can find properties of properties of objects too.

The associativity is left to right.

Refer to the Operator Precedence topic for details of execution order.

You can also access the property values as if the object were an array. This:

```
anObject.aProperty
```

is equivalent to:

```
anObject["aProperty"]
```

The result will be the value of the property when it is an RValue or a reference to the property when it is an LValue.

See also:	Associativity, Decimal point (.), Operator Precedence, Postfix operator

Cross-references:

ECMA 262 edition 2 –section 8.6

ECMA 262 edition 2 –section 11.2

ECMA 262 edition 3 –section 8.6

ECMA 262 edition 3 –section 11.2.1

Wrox *Instant JavaScript* –page 28

OBJECT object (Object/HTML)

This is an object that encapsulates an ActiveX plugin. Do not confuse it with the `Object` object that is the super-class of all objects in JavaScript.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myOBJECT* = *myDocument*.all.*anElementID*
	IE	*myOBJECT* = *myDocument*.all.tags("OBJECT")[*anIndex*]
	IE	*myOBJECT* = *myDocument*.all[*aName*]
	-	*myOBJECT* = *myDocument*.applets[*anIndex*]
	-	*myOBJECT* = *myDocument*.getElementById(*anElementID*)
	-	*myOBJECT* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myOBJECT* = *myDocument*.getElementsByTagName ("OBJECT")[anIndex]
HTML syntax:	`<OBJECT> ... </OBJECT>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	`accessKey, align, altHtml, archive, border, classid, code, codeBase, codeType, data, dataFld, dataSrc, declare, form, height, hspace, name, object, readyState, standby, tabIndex, type, useMap, vspace, width`	
Event handlers:	`onAfterUpdate, onBeforeUpdate, onBlur, onClick, onDataAvailable, onDataSetChanged, onDataSetComplete, onDblClick, onDragStart, onError, onErrorUpdate, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReadyStateChange, onRowEnter, onRowExit, onSelectStart`	

This is an object representing an `<OBJECT>` HTML tag.

The `<OBJECT>` tag is a block-level tag. That means that it forces a line break before and after itself.

This object is specific to the MSIE browser when it runs on the Windows operating system. No other browser supports ActiveX as well as MSIE and no other operating system properly or completely supports the ActiveX infrastructure.

The events handled, and the properties and the methods of this object will depend on the kind of ActiveX object that is created.

The DOM level 1 specification refers to this as an `ObjectElement` object.

Warnings:

- Be very careful not to confuse this object with the generic top level core Object object that is the super-class of all objects in the interpreter.
- This is the object that MSIE instantiates to represent an <OBJECT> tag. MSIE creates OBJECT objects for that purpose but also supports Object objects. For this reason, it may be the case that interpreters cannot become case insensitive when matching class names. If they did, then it would be impossible to distinguish between Object and OBJECT class names.
- Creating an OBJECT class when an Object class already exists must have been a moment of insanity in an otherwise mostly excellent browser implementation project.

See also: ActiveXObject object, Document.applets[], Element object, Input.accessKey, Object object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
altHtml	-	3.0 +	-	4.0 +	-	-	-	-
archive	1.5 +	-	6.0 +	-	-	1 +	-	Deprecated
border	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
classid	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
code	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
codeBase	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
codeType	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
data	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
dataFld	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
dataSrc	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
declare	1.5 +	-	6.0 +	-	-	1 +	-	-
form	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
height	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
hspace	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
name	-	5.5 +	-	5.5 +	-	-	-	-
object	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
readyState	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
standby	1.5 +	-	6.0 +	-	-	1 +	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
type	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
useMap	1.5 +	-	6.0 +	-	-	1 +	-	-
vspace	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onClick	1.5+	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDataAvailable	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetChanged	-	3.0 +	-	4.0 +	-	-	-	-
onDataSetComplete	-	3.0 +	-	4.0 +	-	-	-	-
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onError	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onReadyStateChange	-	3.0 +	-	4.0 +	-	-	-	-
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Web-references:

http://www.w3.org/pub/WWW/TR/WD-object.htm

OBJECT.align (Property)

An alignment control for an <OBJECT> tag's position with respect to its parent object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myOBJECT*.align

The alignment of the ActiveX OBJECT object with respect to its containing parent object is defined in this property. The expected and widely available set of alignment specifiers are available:

- absbottom
- absmiddle
- baseline
- bottom
- center
- left
- middle
- right
- texttop
- top

OBJECT.altHtml (Property)

A block of alternative HTML to be used if the <OBJECT> tag fails to load its plugin correctly.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myOBJECT*.altHtml

The alternative HTML is used in case the base <OBJECT> tag experiences a problem when loading or the browser cannot use ActiveX objects as embeds. Of course if this property is accessible from the OBJECT object during scripting, the browser must have parsed the <OBJECT> tag, although it may still have had problems with the component.

This HTML is enclosed between the <OBJECT> and </OBJECT> tags in the HTML document source.

Certain tags are likely to be omitted from the `altHTML` property value. `<OBJECT>` blocks contain `<PARAMETER>` tags for passing values to the embedded ActiveX component. Clearly you won't want these appearing in the display if the component fails to load. The `<PARAMETER>` tags are considered integral to the `<OBJECT>` and it's smart enough to disregard them as it constructs its alternative HTML block.

Browsers that cannot understand and render `<OBJECT>` tags should also ignore the `<PARAMETER>` tags too.

OBJECT.archive (Property)

A space-separated archive list. This enumerates a set of classes that must be pre-loaded before the object can execute.

Availability:	DOM level –1 JavaScript –1.5 Netscape –6.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myOBJECT*`.archive`

This is a new attribute of the DOM `HTMLObjectElement` but is shown here as this is the existing object type it is to be added to.

OBJECT.border (Property)

The width of the border around the object when it is rendered into the display.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOBJECT*`.border`

This property can be set from script and although its type is a String primitive, it will expect a numeric value. JavaScript will coerce as necessary during the assignment.

OBJECT.classid (Property)

The URL that locates the registered ActiveX control within the local file system when MSIE is used on the Windows platform.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myOBJECT*`.classid`

This is a special URL value used to locate ActiveX objects within the file system of the PC running the client browser. This is likely only available in Windows versions of the MSIE browser because that is the only platform that supports ActiveX objects. It is not supported on the Macintosh version of MSIE because ActiveX objects aren't available there. This is because they are compiled x86 micro-code and therefore cannot run in a non-Intel environment (unless the x86 CPU is being emulated).

The ActiveX control needs to have been registered and installed already. It is possible to construct an `<OBJECT>` tag that conveys sufficient information to locate and install a missing ActiveX control but this can be a quite involved process.

See also:	clsid: URL, URL

Property attributes:

`ReadOnly`.

OBJECT.code (Property)

The name of a Java applet to be used with the `<OBJECT>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOBJECT*`.code`

This specifies the main code class to be loaded when the object is instantiated. This value is added to the codebase property to form a fully qualified URL.

There is conflicting information in the reference sources regarding the read/write ability of this property. Some suggest it is `ReadOnly` and others suggest you can assign a value to it. It may be that you can assign a value to it without the JavaScript interpreter complaining but that any value you assign is ignored.

OBJECT.codeBase (Property)

The path to the directory where the Java applet denoted by the `CLASS="…"` HTML tag attribute is to be found.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOBJECT*`.codebase`

The codebase is the path to the directory where the classes specified in the code property are located. The actual path to the required files is generated by a string concatenation of codebase+code to generate a fully specified URL.

Due to security limitations it is not permitted to access a codebase value that is outside the domain specified by the containing document.

There is conflicting information in the reference sources regarding the read/write ability of this property. Some suggest it is `ReadOnly` and others suggest you can assign a value to it. It may be that you can assign a value to it without the JavaScript interpreter complaining but that any value you assign is ignored.

OBJECT.codeType (Property)

A description of the type of code in the object referred to by the `CLASSID="…"` HTML tag attribute.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOBJECT*`.codeType`

This is a MIME type value that describes the kind of code being embedded by the `<OBJECT>` tag.

There is conflicting information in the reference sources regarding the read/write ability of this property. Some suggest it is `ReadOnly` and other suggest you can assign a value to it. It may be that you can assign a value to it without the JavaScript interpreter complaining but that any value you assign is ignored.

See also:	MIME types

OBJECT.data (Property)

A URL that points at a file containing data that the `OBJECT` element can access.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOBJECT*`.data`

This is intended for passing in a URL that the ActiveX object can use to access some data service that is online and available for access via the network. It is not the URL of the ActiveX object itself.

Property attributes:

`ReadOnly`.

OBJECT.declare (Property)

A means of defining the object without activating it.

Availability:	DOM level –1 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myOBJECT*`.declare`

Declaring an `OBJECT` in this way may be useful when referring to the object from elsewhere in the page or from within another object. Sometimes you simply want to know something about it, perhaps one of its parameters. For video players, sometimes its useful to instantiate the `OBJECT` into the display without playing the video right away.

OBJECT.form (Property)

The form that an object belongs to if it is used for form input.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Form` object	
JavaScript syntax:	-	*myOBJECT*`.form`

Property attributes:

`ReadOnly`.

OBJECT.height (Property)

The height of an area reserved for displaying the contents of the `<OBJECT>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myOBJECT*`.height`

The object space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is the smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

OBJECT.hspace (Property)

A horizontal margin space either side of the `<OBJECT>` tag with respect to its surrounding objects.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myOBJECT*`.hspace`

Margins placed around objects are either modified separately with all four margin sides having a different property or by adjusting the horizontal margins and vertical margins using just two values.

The `hspace` property controls the margin to the left and right of the object.

OBJECT.name (Property)

The value of the `NAME="…"` HTML tag attribute.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOBJECT*`.name`

Objects are identified either by the `NAME="…"` HTML tag attribute or by the `ID="…"` HTML tag attribute.

Netscape shows a marginal preference for the `name` property while MSIE seems slightly better disposed towards the `ID` property. However in many cases, both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

See also:	`NAME="…"`, Namespace

OBJECT.object (Property)

An accessor that yields a reference to the containing JavaScript object when there is a possibility of naming conflicts between internally visible and externally visible property names.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Object` object	
JavaScript syntax:	IE	*myOBJECT*`.object`

If a property is a public property of the ActiveX object, and that name coincides with a property of the JavaScript object that is instantiated by the `<OBJECT>` HTML tag, then access to the property belonging to the containing object is difficult. This is because the search order will see the public property of the ActiveX object first. By using the `object` property, once can access the containing object explicitly and retrieve a property of that object even if there is an identically named property belonging to the enclosed ActiveX object.

This access mechanism applies to method invocations as well.

See also:	`Applet.object`

Property attributes:

`ReadOnly`.

OBJECT.readyState (Property)

The current status disposition of the <OBJECT> tag as it is being loaded.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myOBJECT*.readyState

This property reflects the loading status of an <OBJECT> tag and its corresponding OBJECT object instantiation.

Sometimes, you can design scripts to execute while the document is downloading. Inline scripts for example. At that time, you may even be able to trigger interval timed deferred executions as well.

If it is important that the document has completed loading, you can check this property for one of the following values:

State	Value
uninitialized	The object is first instantiated but has not begun loading.
loading	The object has commenced loading.
loaded	The object has completed loading.
interactive	The object is loaded but not yet closed but is ready to handle interaction.
complete	The object body has been closed and the loading is finished.

An object may not need to reflect the complete status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an OBJECT object that represents an <OBJECT> tag until the <BODY> has completed loading. This is because the ActiveX object construction requires a complete document body structure to attach itself to.

Every time this readyState value changes, it triggers an onReadyStateChange event call-back.

See also:	onReadyStateChange

Property attributes:

ReadOnly.

OBJECT.standby (Property)

Sets or resets the message text displayed while the object is loading.

Availability:	DOM level –1 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myOBJECT*`.standby`

OBJECT.tabIndex (Property)

A control of where the `OBJECT` object appears in the tabbing order of the page.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myOBJECT*`.tabIndex`

This value indicates where in the tabbing sequence this object and any of its children will be placed. The tabbing order is used when filling in forms or moving focus. Pressing the `[`*tab*`]` key moves from one form element to the next according to the cascaded tabbing order defined by building a tree-like structure with the tab index values.

OBJECT.type (Property)

An indication of the MIME type of the object if its `codeType` property is undefined.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOBJECT*`.type`

The MIME type of the object is accessible through the value of this property.

Refer to the MIME type topic for details of the available MIME types you will likely see in this property.

See also:	MIME types

OBJECT.useMap (Property)

The URL of a <MAP> defined hash element that defines a client-side image map.

Availability:	DOM level –1 JavaScript –1.5 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	N	*myOBJECT*.useMap

This property reflects the value of the USEMAP="…" HTML tag attribute, which should refer to the named <MAP> tag containing an image map. The reference is by means of a "#NAME" value in this property that corresponds to the NAME="…" HTML tag attribute of the <MAP> tag describing the image map to use with the object.

OBJECT.vspace (Property)

A vertical spacing above and below the <OBJECT> with respect to its adjacent objects.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myOBJECT*.vspace

Margins placed around objects are either modified separately with all four margin sides having a different property or by adjusting the horizontal margins and vertical margins using just two values.

The vspace property controls the margin at the top and bottom of the object.

OBJECT.width (Property)

The height of an area reserved for displaying the contents of the <OBJECT> tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myOBJECT*.width

The object space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is the smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

Obsolescent (Definition)

A feature of the language that is no longer supported.

Warnings:

- If you use obsolescent functionality, your script may fail when it is deployed on other platforms.

See also:	Deprecated functionality

Octal value (Definition)

A numeric value based on a radix of 8.

Availability:	ECMAScript edition –3

An octal value is an integer composed of only the following characters:

0 1 2 3 4 5 6 7

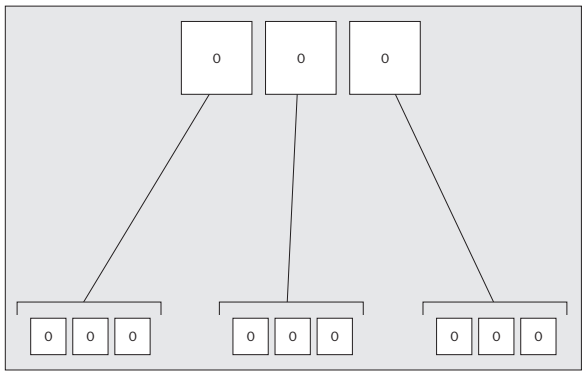
Octal values are always prefixed by a zero character.

The sequence carries over for the next increment when each column reaches the value 7. Thus:

00 01 02 03 04 05 06 07 010 011 012

Octal values have a historical significance from having been used in the earliest computer systems. However these days, they are particularly useful since they map quite conveniently to the binary system. Each octal digit corresponds to three binary digits.

The most significant of the three octal digits does not have a full range since it contains a carry over bit and a three digit octal number actually represents a 9 bit value. However, an 8 bit value can be encoded conveniently if the range is limited to 0377 as a maximum. Hexadecimal values map far more conveniently although they are harder to compute mentally.



Warnings:

- Beware when you prefix decimal values with a zero character. You may want to justify a column of figures. If you add leading zero characters to a numeric string, if that string is subsequently parsed back to a numeric value, you may inadvertently export the value as a decimal but import it as an octal value. This can lead to an extremely difficult-to-diagnose fault in your software because the parsers sometimes add some intelligence and will correctly interpret the value as decimal if the characters 8 or 9 are present, but otherwise interpret it as octal notation.
- This may be implementation-dependant behavior to some extent.
- Be careful that you remove any leading zero characters from the text strings that you plan to convert using the numeric parser. The example shows a simple function for doing this.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myString = "00123";
document.write(stripLeadingZeros(myString));
// Strip leading zero characters off a numeric string
function stripLeadingZeros(aString)
{
   return aString.substr(aString.search(/[1-9]/));
}
</SCRIPT>
</BODY>
</HTML>
```

See also: Decimal value, Hexadecimal value, Integer constant, Number, `Number.toString()`

Cross-references:

ECMA 262 edition 3 –section –B.1

O'Reilly *JavaScript Definitive Guide* –page –35

Off by one errors (Pitfall)

An error caused by missing the target value by one.

This kind of errors are caused by the following:

- Forgetting than an index is zero-based and assuming it begins at 1. This typically affects arrays and strings.
- Enumerating through a range of values and testing for equality with the target value rather than testing that you are still less than the target value. This is typically a problem when you build for loops.

See also: Array index delimiter ([ ]), `Array.slice()`, `do ... while( ... )`, `for( ... ) ...`, Pitfalls, `while( ... ) ...`

Off-screen image caching (Useful tip)

A technique for caching images locally in readiness for an animation.

Refer to:

Image preloading

offscreenBuffering (Property)

An alias for the `window.offScreenBuffering` property.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Property/method value type:	`Boolean` or `String primitive`	
JavaScript syntax:	-	*myWindow*`.offScreenBuffering`
	-	*myWindow*`.offScreenBuffering = `*aSetting*
	-	`offScreenBuffering`
	-	`offScreenBuffering = `*aSetting*
Argument list:	*aSetting*	A new value to control this functionality

Refer to:

`Window.offScreenBuffering`

OL object (Object/HTML)

An object that represents the ordered list contained in an `<OL>` tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`myOL = `*myDocument*`.all.`*anElementID*
	IE	*myOL* `= `*myDocument*`.all.tags("OL")[`*anIndex*`]`
	IE	*myOL* `= `*myDocument*`.all[`*aName*`]`
	-	*myOL* `= `*myDocument*`.getElementById (`*anElementID*`)`
	-	*myOL* `= `*myDocument*`.getElementsByName(`*aName*`)[`*anIndex*`]`
	-	*myOL* `= `*myDocument*`.getElementsByTagName ("OL")[`*anIndex*`]`

HTML syntax:	`<OL> ... </OL>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`compact, start, type`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

The `<OL>` tag is a block-level tag. That means that it forces a line break before and after itself.

The DOM level 1 standard describes this as a `HTMLOListElement` object.

See also:	Element object, `UL` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`compact`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`start`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`type`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

OL.compact (Property)

A switching attribute that condenses the space required to display the ordered list on the screen.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myOL*`.compact`

The collection of `LI` objects are presented in the normal spaced out style when the compact property belonging to their owner `OL` object is set to `false`.

Setting the property to `true` should result in the list items being squeezed closer together. however the functionality is rarely supported on web browsers.

Its more likely that you'll apply CSS style attributes to the list to achieve the same effect.

Warnings:

- Setting the property in MSIE and Netscape is quietly ignored by both browsers. No visible effect, no error message.

See also:	`LI` object

OL.start (Property)

The starting index of items in the ordered list. The enumerator can be set to a predetermined value with this property.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myOL*`.start`

Ordered lists can start at any value. This is somewhat related to the `LI.value` property, which allows the list items to begin sequencing from any value you care to define. The value you specify here must be a positive integer.

Warnings:

- This exhibits some bugs in Netscape 6.0. Setting a start value seems to be applied inconsistently to the items in the list. They are renumbered but not correctly. For now, the best work around is to construct the list and use the `innerHTML` trick to store it into the page. This will likely get fixed as soon as people really get to grips with Netscape 6.0 and it does work fine in MSIE.

See also:	LI.value

OL.type (Property)

The presentation style of the ordered list.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOL*`.type`

Although the sequence numbers are incrementing in an ordered list, they may be displayed in a variety of different formats selected by this property.

You can override this on an item-by-item basis and so this property is related to the `LI.type` property.

The following type values are appropriate for this list type:

- `1` - Numeric
- `a` –Alphabetical –lower case
- `A` –Alphabetical –upper case
- `i` –Roman numerals –lower case
- `I` –Roman numerals –upper case

See also:	`LI.type, style.listStyleType, UL.type`

on ... (Event handler)

All event handlers begin with the word on.

The event handlers are many and various in their support across the browsers and in the way that they are triggered. There is no one object that supports them all and even when an object does not support them by default, the event-management capabilities of MSIE or Netscape can sometimes set objects up to support them anyway.

Here is a summary of the available supported events:

Event	Handler	Usage
Abort	`onabort`	When image loading is aborted.
AfterPrint	`onafterprint`	When printing has just finished.
AfterUpdate	`onafterupdate`	When an update is completed.
Back	`onback`	The user has clicked on the `[BACK]` button in the toolbar.
BeforeCopy	`onbeforecopy`	Immediately before a copy to the clipboard.
BeforeCut	`onbeforecut`	Immediately before a cut to the clipboard.
BeforeEditFocus	`onbeforeeditfocus`	Immediately before the edit focus is directed to an element.
BeforePaste	`onbeforepaste`	Immediately before the clipboard is pasted.
BeforePrint	`onbeforeprint`	Immediately before printing begins.
BeforeUnload	`onbeforeunload`	Called immediately prior to the window being unloaded.
BeforeUpdate	`onbeforeupdate`	Called immediately before an update commences.
Blur	`onblur`	When an input element loses input focus.
Bounce	`onbounce`	Triggered when a marquee element hits the edge of its element area.
Change	`onchange`	When edit fields have new values entered or a popup has a new selection, this event's handler can check the new value.
Click	`onclick`	When the user clicks the mouse button on the `Element` object that represents the object on screen.
ContextMenu	`oncontextmenu`	Special handling for contextual menus.
Copy	`oncopy`	When a copy operation happens.
Cut	`oncut`	When a cut operation happens.
DataAvailable	`ondataavailable`	Some data has arrived asynchronously from an applet or data source.
DataSetChanged	`ondatasetchanged`	A data source has changed the content or some initial data is now ready for collection.
DataSetComplete	`ondatasetcomplete`	There is no more data to be transmitted from the data source.
DblClick	`ondblclick`	When the user double-clicks on an object.
Drag	`ondrag`	When a drag operation happens.
DragDrop	`ondragdrop`	Some data has been dropped onto a window.
DragEnd	`ondragend`	When a drag ends.
DragEnter	`ondragenter`	When a dragged item enters the element.
DragLeave	`ondragleave`	When a dragged item leaves the element.
DragOver	`ondragover`	While the dragged item is over the element.
DragStart	`ondragstart`	The user has commenced some data selection with a mouse drag.
Drop	`ondrop`	When a dragged item is dropped.
Error	`onerror`	Triggered if an error occurs when loading an image.

Table continued on following page

Event	Handler	Usage
ErrorUpdate	`onerrorupdate`	An error has occurred in the transfer of some data from a data source.
FilterChange	`onfilterchange`	A filter has changed the state of an element or a transition has just been completed.
Finish	`onfinish`	A marquee object has finished looping.
Focus	`onfocus`	When the form element is selected for entry.
Forward	`onforward`	The user has clicked on the `[FORWARD]` button in the toolbar.
Help	`onhelp`	The user has pressed the `[F1]` key or selected `[help]` from the toolbar or menu.
KeyDown	`onkeydown`	When a key is pressed.
KeyPress	`onkeypress`	Pressing the key down and releasing it again elicits this event.
KeyUp	`onkeyup`	When a key is released.
Load	`onload`	When an object has completed loading.
LoseCapture	`onlosecapture`	When an element loses event capturing permission.
MouseDown	`onmousedown`	When the mouse button is pressed.
MouseDrag	`onmousedrag`	An event handler for mouse drag operations.
MouseMove	`onmousemove`	When the mouse pointer is moved.
MouseOut	`onmouseout`	When the mouse pointer leaves the active area occupied by the `Element` object that represents the object on screen.
MouseOver	`onmouseover`	When the mouse pointer enters the active area owned by the object.
MouseUp	`onmouseup`	When the mouse button is released.
Move	`onmove`	The browser window has been moved..
Paste	`onpaste`	When a paste operation happens.
PropertyChange	`onpropertychange`	When an object property is modified (similar to the Netscape `watch()` method).
ReadyStateChange	`onreadystatechange`	An object in the window has changed its ready state.
Reset	`onreset`	The user has clicked a reset button in a form.
Resize	`onresize`	As the window is resized, this event is triggered.
RowEnter	`onrowenter`	The data in a field bound to a data source is about to be changed.
RowExit	`onrowexit`	The data in a field bound to a data source has been changed
Scroll	`onscroll`	The window has been scrolled.
Select	`onselect`	Some textual content in the window has been selected.
SelectStart	`onselectstart`	A select action is beginning.
Start	`onstart`	A marquee element is beginning its loop.
Stop	`onstop`	When a stop action occurs.
Submit	`onsubmit`	The user has clicked on the submit button in a form
Unload	`onunload`	Triggered when the document is unloaded.

Events are associated with HTML tags by means of the `onEvent="..."` tag attribute. This assigns a function object to the onevent property of the `Element` object that represents a tag. Event handlers can be associated with objects by assigning function object references to the event-handler properties of the objects. However, this is also not consistently supported across the browsers.

Netscape supports event routing calls that can send events to objects by passing the `Event` object to them. The objects then map the events to an appropriate handler according to their set-up.

The initial value of this property will be an anonymous function whose body contains the contents of the `onEvent="..."` HTML tag attribute.

When the event is triggered and called, it will execute in the context of the `Element` object and not the window. This means the execution scope will be that of the `Element` object.

In Netscape prior to version 6.0, event handlers are passed an event object as an argument when they are called. In MSIE, no object is passed but the equivalent event object is available as the event property of the window that contains the `Element` object that receives the event trigger.

On the return from a handler, other processing may take place. In pre-version 6.0 Netscape browsers, the general technique for this is to return a `false` value. Returning nothing is equivalent to returning `true` which may allow some default processing in the browser to be called. In MSIE, this flag value is also supported, at least on later browser versions. In addition, the `returnValue` property of the event object can be set to `false`.

Now with the version 6.0 release of Netscape the DOM level 2 event model has been introduced. The event model is still undergoing some development at the W3C standards organisation and a level 3 update is available for review. This at least is a turning point for event handling because all browser manufacturers have stated that it is their goal to be standards-compliant. The new event model is mainly based on the MSIE way of doing things but it is a slightly hybrid and takes some ideas from the Netscape tradition too.

Warnings:

- Prior to version 6.0 of Netscape, anything more than simple event handling is managed differently between the Netscape and MSIE browsers.
- Properties that can be assigned with a function handler vary between Netscape and MSIE. Generally, MSIE allows this property assigning technique more completely and consistently across all its event handlers and objects that own them.
- Be aware that you don't always get a meaningful `event` object passed as an argument to the event handler in Netscape. On some platforms, the `event` object may have some useful properties, on others it may have none. The MSIE `event` object is more complete but less flexible as it needs to be shared between all events.
- Event handling support has radically changed in Netscape 6.0, and because it is a new code base and a new standard, there are likely to be bugs and limitations in its support for a while yet.

See also:	`Element.onevent`, `Event` object

onAbort (Event handler)

An event that happens when loading is interrupted.

Availability:	JavaScript –1.1 JScript –1.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onabort = *aHandler*
HTML syntax:	`<HTMLTag onAbort="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`IMG`	

This event is only supported by `IMG` objects. It is triggered if a page load is aborted partway through the loading of an image.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onError, onLoad`, Semantic event

onAfterPrint (Event handler)

Called when printing is completed.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*myObject*.onafterprint = *aHandler*

This provides a hook for cleaning up after printing a page.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onBeforePrint`, Semantic event

onAfterUpdate (Event handler)

An event that happens after an update.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onafterupdate = `*aHandler*
HTML syntax:	`<HTMLTag onAfterUpdate="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`APPLET, Checkbox, Input, TABLE, AREA, DIV, MARQUEE,TD, BODY, Document, OBJECT, TEXTAREA, BUTTON, FIELDSET, RadioButton, TH, CAPTION, IMG, Select`	

If the contents of a page are database driven, this will be called after the database has been updated.

See also:	Event, Event handler, Event model, Event names, Event object, Event.returnValue, Handler, onBeforeUpdate, Semantic event

onBack (Event handler)

Triggered by the back button.

Availability:	JavaScript –1.3 Netscape –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myObject*`.onback = `*aHandler*
HTML syntax:	`<HTMLTag onBack="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event

This event is called when the user interacts with the browser. Some events like this are helpful for blocking the operation of certain browser UI elements.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onForward`, Semantic event, Window events

onBeforeCopy (Event handler)

Called immediately before a copy operation.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.onbeforecopy = aHandler`

If a selected area of the page is about to be copied to the clipboard, this allows the event to be intercepted and blocked if necessary.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onBeforeCut`, `onBeforePaste`, Semantic event

onBeforeCut (Event handler)

Called immediately before a cut operation.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.onbeforecut = aHandler`

If a selected area of the page is about to be cut to the clipboard, this allows the event to be intercepted and blocked if necessary.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onBeforeCopy`, `onBeforePaste`, Semantic event

onBeforeEditFocus (Event handler)

Called immediately before focus is relocated to another object.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.onbeforeeditfocus = aHandler`

If a field within a form is about to be edited, this allows the event to be intercepted and blocked if necessary.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Semantic event

onBeforePaste (Event handler)

Called immediately before a paste operation.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.onbeforepaste = aHandler`

If a form elements is about to have its contents pasted from the clipboard, this allows the event to be intercepted and blocked if necessary.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onBeforeCopy`, `onBeforeCut`, Semantic event

onBeforePrint (Event handler)

Called immediately before printing commences.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.onbeforeprint = aHandler`

This provides a hook to set the page up before printing. This may be necessary to change the content of some items in the page so that they print more attractively.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onAfterPrint`, Semantic event

onBeforeUnload (Event handler)

An event before an unload happens.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`myObject.onbeforeunload = aHandler`
HTML syntax:	`<HTMLTag onBeforeUnload="...">`	
Argument list:	`aHandler`	A reference to a function object to handle the event
Supported by objects:	BODY, Layer, SCRIPT, Window FRAMESET, LINK, STYLE,	

This event is triggered immediately before an unload event. It provides a hook for cleaning up immediately before a page is unloaded.

It is bad User Interface design to call an alert box at this point in the proceedings.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onUnload`, Semantic event

onBeforeUpdate (Event handler)

An event called before an update happens.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onbeforeupdate = `*aHandler*
HTML syntax:	`<HTMLTag onBeforeUpdate="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`APPLET, Checkbox, Input, TD,AREA, DIV, OBJECT, TEXTAREA, BODY, Document, RadioButton, TH, BUTTON, FIELDSET, Select, CAPTION, IMG, TABLE,`	

Database updates my require some data-integrity checks to be carried out. This event trigger allows some checking to take place at an opportune moment, immedately before the update happens.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onAfterUpdate`, Semantic event

onBlur (Event handler)

Triggered when the user selects another form element for input and the current one loses focus.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –3.0 Netscape –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*`.onblur = `*aHandler*
HTML syntax:	`<BODY onBlur="`*aHandler*`"> <FRAMESET onBlur="`*aHandler*`"> <HTMLTag onBlur="`*aHandler*`"> <INPUT onBlur="`*aHandler*`">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.

Supported by objects:	A, Embed, OBJECT, TD, Anchor, FIELDSET, Password, TEXTAREA, APPLET, FileUpload, RadioButton, TextCell, AREA, IMG, ResetButton, TH, BUTTON, Input, Select, TR, CAPTION, Layer, SPAN, Url, Checkbox, LEGEND, SubmitButton, DIV, MARQUEE, TABLE,

A blur event is caused by the user clicking on another window or frame or the `blur()` method being called for an object. When this event is triggered, an `onBlur` event handler will be invoked.

The `onblur` event handler is a function which is represented by an object that is referred to by this property. Because it is stored in a property, you can change the handler by storing a reference to a different function object in this property. At least, you can on MSIE.

You cannot redefine the value of the `window.onblur` property from inside the `onblur` function handler. This means you can't modify the `onblur` behavior while a blur event is in progress.

Netscape will pass an `event` object as an argument when it calls this event handler function. MSIE does not pass an object but makes the event data available via the `Event` object that is stored and accessed globally for all events.

DOM level 2 events refers to this as a `DOMFocusOut` event, which employs event bubbling for its propagation and cannot be canceled.

Warnings:

- It is somewhat easy to create an `onBlur/onFocus` recursion which leads to an endless loop with objects exchanging focus backwards and forwards. Focus on *`object A`* triggers a blur event on *`object B`*, which tries to wrest back the focus again because it believes its content is incomplete. Meanwhile *`object A`* isn't happy with its content either and so they play 'Tug-of-War' with one another. This continues until the browser has a race hazard attack and GPFs –that is if it hasn't already fallen over due to a stack overflow. If you're lucky only the browser will crash, but it could bring down the OS as well.

See also:	Event, Event handler, Event model, Event names, Event object, Event.returnValue, Handler, Input.blur(), Input.focus(), onFocus, Password object, Semantic event, UIEvent object, Window.onblur

onBounce (Event handler)

Triggered when a marquee element hits the edge of its element area.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myObject`*`.onbounce = `*`aHandler`
HTML syntax:	`<HTMLTag onBounce="...">`	
Argument list:	*`aHandler`*	A reference to a function object to handle the event
Supported by objects:	`MARQUEE`	

You may want to change the MARQUEE content or do some wizzy animation effect when the bounce occurs.

Now that we can do pretty much the same things with dynamic HTML and style-driven positioning, this could become deprecated.

Its still quite useful though, because it knows how big the marquee content is and it triggers the bounces automatically. That's possible with CSS scripting but its not completely trivial because you need to measure the extent rectangle of an object, which could involve some very messy font metric calculations.

See also:	Event, Event handler, Event model, Event names, Event object, Event.returnValue, Handler, onStart, onStop, Semantic event

onChange (Event handler)

Triggered when the value belonging to an input element is changed.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0 Opera browser –3.0
Property/method value type:	Boolean primitive
JavaScript syntax:	- *myObject*.onchange = *aHandler*
HTML syntax:	<HTMLTag onChange="..."> <INPUT onChange="...">
Argument list:	*aHandler* A reference to a function object to handle the event
Supported by objects:	BODY, FIELDSET, LEGEND, TEXTAREA, CAPTION, FileUpload, Password, TextCell, Checkbox, IMG, RadioButton, DIV, Input, Select,

When the data in the input element is modified, the ONCHANGE event handler is called.

You can place some script code directly into this handler like this:

```
<INPUT TYPE="TEXT" ONCHANGE="alert('Changed');">
```

In this context, you can refer to the input element itself using the 'this' keyword. Thus:

```
<INPUT TYPE="TEXT" ONCHANGE="this.value=checkValue(this.value);">
```

This will check the value and reset it to an allowed value. That may be to force it to be all lower case for example or to remove weird characters.

Actually, you don't need to pass *this.value* because it's accessible from within the function. Functions inherit some scope according to how they are called so this would be better:

```
<INPUT TYPE="TEXT" ONCHANGE="this.value=checkValue();">

<SCRIPT>

function checkValue(anObject)

{

this.value = this.value + "XXX";

return true;

}

</SCRIPT>
```

This is also a particularly useful event to handle on behalf of a `<SELECT>/<OPTION>` control although it is not supported fully in all browsers.

Because the MSIE version 3.0 browser doesn't trigger this event, it has become popular to place a small input button beside the selector so the user can select and then trigger. If you do that, you really should make that technique browser-version sensitive and 'do the automatic thing' when you can.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, `FileUpload` object, Handler, `Password` object, `Select` object, Semantic event, `TEXTAREA` object, `TextCell` object

Cross-references:

Wrox *Instant JavaScript* –page –53

onClick (Event handler)

This event is triggered when the user clicks the mouse button with the pointer over the `Element` object that represents the object on screen.

Availability:	HTML version –4.0 JavaScript –1.0 JScript –1.0 Internet Explorer –3.0 Netscape –2.0 Opera browser –3.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *myObject*.onclick = *aHandler*
HTML syntax:	`<A onClick="..."> <AREA onClick="..."> <HTMLTag onClick="..."> <INPUT onClick="...">`

Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	A, Checkbox, FileUpload, LISTING, S, TEXTAREA, ACRONYM, CITE, FONT, MAP, SAMP, TextCell, ADDRESS, CODE, FORM, MARQUEE, Select, TFOOT Anchor, DD, H1, MENU, SMALL, TH, APPLET, DEL, HR, OBJECT, SPAN, THEAD, AREA, DFN, I, OL, STRIKE, TR, B, DIR, IMG, P, STRONG, TT, BIG, DIV, Input, Password, SUB, U, BLOCKQUOTE, DL, INS, PLAINTEXT, SubmitButton, UL, BODY, Document, KBD, PRE, SUP, Url, BUTTON, DT, LABEL, Q, TABLE, VAR, CAPTION, EM, LEGEND, RadioButton, TBODY, CENTER, FIELDSET, LI, ResetButton, TD,		

The `onClick` event handler is invoked when the user clicks once on the object that it belongs to. This might normally be a hypertext link. As of version 1.1 of JavaScript, if the handler returns the Boolean false value then the browser will not follow the link to its HREF. If a true value is returned then it will.

This event applies to anchors, reset buttons, and submit buttons. A `false` return value inhibits the browser from taking any default actions once your handler is completed.

Netscape indicates which mouse button was pressed in the `which` property of the `Event` object that is passed as an argument to the event handler function. MSIE makes the value available in the `button` property of the `event` object referenced by the `window.event` property.

DOM level 2 refers to this as a `DOMActivate` event, which employs event bubbling for its propagation and can be canceled. The context info provides detail about whether its was a single or double click.

DOM level 2 also classifies this as a `MouseEvent` and specifies that it must follow a matching pair of `mousedown` and `mouseup` events without any intervening mouse movement. As a `MouseEvent` it uses bubbling propagation and can be canceled.

Warnings:

- When you add an `onClick` handler to an `<A>` tag object, MSIE and Netscape execute the click in the same way but modify the browser history differently.
- An `onClick` event will also generate an `onMouseDown` event and an `onMouseUp` event. The `onClick` will not be triggered until the `onMouseUp` event. There is no guarantee that the `onClick` event will arrive after the `onMouseUp` event although that is the logical order.

See also:	`Button` object, `BUTTON` object, `Checkbox` object, `Element` object, `Element.click()`, Event, Event handler, Event model, Event names, `Event` object, `Event.button`, `Event.returnValue`, `Event.which`, Handler, `Input.click()`, Mouse events, `MouseEvent` object, `RadioButton` object, `ResetButton` object, Semantic event, `SubmitButton` object, `UIEvent` object

onContentReady (Event handler)

A special event handler provided to facilitate the loading of behavior controls.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.oncontentready = aHandler`

This event is notified to the behavior handler when the content of the element that the behavior is associated with has been loaded and parsed. This does not mean that the document is complete yet. That is signified by the `onDocumentReady` event being notified to the behavior script.

See also:	`onDocumentReady`

onContextMenu (Event handler)

Called when a context menu is requested by the user.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.oncontextmenu = aHandler`

Contextual menus can trigger this event as an item is selected. This allows you you to activate some context specific code at that time.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Semantic event

onCopy (Event handler)

Called when a copy operation is requested by the user.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.oncopy = aHandler`

If the user copies some page content to the clipboard, you may want to know that it has happened and do something about it or perhaps even block the action.

See also:	`dataTransfer.getData()`, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onCut`, `onPaste`, Semantic event

onCut (Event handler)

Called when a Cut operation is requested by the user.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*myObject*.oncut = *aHandler*

If the user cuts some page content to the clipboard, you may want to know that it has happened and do something about it or perhaps even block the action.

See also:	dataTransfer.getData(), Event, Event handler, Event model, Event names, Event object, Event.returnValue, Handler, onCopy, onPaste, Semantic event

onDataAvailable (Event handler)

Some data has arrived asynchronously from an applet or data source.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	IE	*myObject*.ondataavailable = *aHandler*
HTML syntax:	<HTMLTag onDataAvailable="...">	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	APPLET, AREA, BODY, IMG, OBJECT	

This event fires periodically as data arrives from data source objects that asynchronously transmit their data.

In a data-driven system, occasionally you will need to wait for the arrival of some data. This event triggers when the data has arrived, and you can build a script that works with the data that is triggered from this event, rather than build some polling code that sits there busily waiting for things to arrive.

See also:	Event, Event handler, Event model, Event names, Event object, Event.returnValue, Handler, onDataSetChanged, onDataSetComplete, Semantic event, XML.event

onDataSetChanged (Event handler)

A data source has changed the content or some initial data is now ready for collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.ondatasetchanged = `*aHandler*
HTML syntax:	`<HTMLTag onDataSetChanged="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`APPLET, AREA, BODY, IMG, OBJECT`	

This event fires when the data set exposed by a data source object changes.

When sharing data out of a database, it is possible that there are contentions or users trying to access and change the same data. You may need to watch a piece of data and perform some operation when it changes. This trigger is activated when that happens.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDataAvailable`, Semantic event, `XML.event`

onDataSetComplete (Event handler)

There is no more data to be transmitted from the data source.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.ondatasetcomplete = `*aHandler*
HTML syntax:	`<HTMLTag onDataSetComplete="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`APPLET, AREA, BODY, IMG, OBJECT`	

This event fires to indicate that all data is available from the data source object.

In a data-driven loop, you may want to perform some processing at closure when the data has been completely retrieved from the database. This event is triggered when that happens.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.reason`, `Event.returnValue`, Handler, `onDataAvailable`, Semantic event, `XML.event`

onDblClick (Event handler)

Triggered when the user double-clicks on an object.

Availability:	HTML version –4.0 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.ondblclick = *aHandler*
HTML syntax:	`<A onDblClick="..."> <AREA onDblClick="..."> <BODY onDblClick="..."> <HTMLTag onDblClick="..."> <IMG onDblClick="..."> <INPUT onDblClick="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`A, DIR, LI, STRIKE,ACRONYM, DIV, LISTING, STRONG,ADDRESS, DL, MAP, SUB,Anchor, Document, MARQUEE, SubmitButton,APPLET, DT, MENU, SUP,AREA, EM, OBJECT, TABLE,B, FIELDSET, OL, TBODY,BIG, FileUpload, P, TD,BLOCKQUOTE, FONT, Password, TEXTAREA,BODY, FORM, PLAINTEXT, TextCell,BUTTON, H1, PRE, TFOOT,CAPTION, HR, Q, TH,CENTER, I, RadioButton, THEAD,Checkbox, IMG, ResetButton, TR,CITE, Input, S, TT,CODE, INS, SAMP, U,DD, KBD, Select, UL,DEL, LABEL, SMALL, Url,DFN, LEGEND, SPAN, VAR,`	

This event is generated when the user clicks on the receiving object twice.

Netscape indicates which mouse button was pressed in the `which` property of the `event` object that is passed as an argument to the event handler function. MSIE makes the value available in the `button` property of the event object referenced by the `window.event` property.

Warnings:

- This is not supported by Netscape 4.0 on Unix or Macintosh platforms.
- An `onDblClick` event will also trigger `onMouseDown` and `onMouseUp` events. There will also be a possibility of `onClick` event triggering in some implementations, although a single click and double click should be distinguishable from one another. There is no guarantee that the `onDblClick` will arrive after the second `onMouseUp` although that is the logical order.

See also:	Element object, Event, Event handler, Event model, Event names, `Event` object, `Event.button`, `Event.returnValue`, `Event.which`, Handler, Mouse events, `onClick`, `onMouseDown`, `onMouseUp`, Semantic event

onDocumentReady (Event handler)

A special event to signify that a document is loaded and ready for use.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*myObject*`.ondocumentready = `*aHandler*

This event is notified to the behavior handler when the document has been downloaded and completely parsed. The handler needs to be able to distinguish between different events that may trigger it and lock out any user interaction until after this notification has been received.

See also:	`onContentReady`

onDrag (Event handler)

Called when a Drag is activated by the user.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*myObject*`.ondrag = `*aHandler*

Drag and drop code is notoriously non-portable and hard to manage. This family of event triggers greatly facilitate that process although they are not widely supported on all browsers.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDragDrop`, `onDragEnd`, `onDragEnter`, `onDragLeave`, `onDragOver`, `onDragStart`, `onDrop`, Semantic event

onDragDrop (Event handler)

Some data has been dropped onto a window.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*`.ondragdrop = `*aHandler*

HTML syntax:	<BODY onDragDrop="*aHandler*"> <FRAMESET onDragDrop="*aHandler*"> <HTMLTag onDragDrop="*aHandler*">	
Argument list:	*aHandler*	A reference to a function object to handle the event

This is triggered when some data is dropped onto an object in the window. You can then access the data and decide what to do with it.

This event handler is most likely to be invoked when the user drags an item onto a window in Netscape.

To access the details of the entity that has been dragged into and dropped on the window, you need to inspect the `data` property of the `event` object that is passed as an argument to the handler when it is called.

The data is a single URL when a single entity is dropped into the window or an array of strings, each containing a URL when a collection of entities are dropped onto a window. What you can then do with those entity references really depends on your platform and browser capabilities and what you can do to files on your client system given the security implications of browser access to the filesystem.

You will need `UniversalBrowserRead` privilege to access this data in Netscape.

The handler is registered either by assigning a function to the `ondragdrop` property or by defining it with an HTML tag attribute.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDrag`, `onDragEnd`, `onDragEnter`, `onDragLeave`, `onDragOver`, `onDragStart`, `onDrop`, Semantic event, `UniversalBrowserAccess`, `UniversalBrowserRead`, `Window.ondragdrop`

onDragEnd (Event handler)

Called when the drag finishes and the dragged object is dropped and released.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*myObject*.ondragend = *aHandler*

This provides a more fine-grain approach to drag-drop management. It may be that you want to display some active animated effect while the drag is in progress. This tells you the dragging is now finished.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDrag`, `onDragDrop`, `onDragEnter`, `onDragLeave`, `onDragOver`, `onDragStart`, `onDrop`, Semantic event

onDragEnter (Event handler)

Called when a dragged object enters the receiving object.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.ondragenter = aHandler`

Drag and drop may be fine-tuned to apply to only parts of the page. With this, you can do some artful rollover effects to highlight locations on the page where drags and drop is OK and where its not.

See also:	`dataTransfer.dropEffect`, Event, Event handler, Event model, Event names, Event object, `Event.returnValue`, Handler, `onDrag`, `onDragDrop`, `onDragEnd`, `onDragLeave`, `onDragOver`, `onDragStart`, `onDrop`, Semantic event

onDragLeave (Event handler)

Called when a dragged object leaves the receiving object.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.ondragleave = aHandler`

Rolling off of a dragged 'hot' item triggers this event. This is like a rollover effect but it is active only while dragging.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDrag`, `onDragDrop`, `onDragEnd`, `onDragEnter`, `onDragOver`, `onDragStart`, `onDrop`, Semantic event

onDragOver (Event handler)

Called repeatedly while the dragged object is over the receiving object.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	`myObject.ondragover = aHandler`

Drag over effects to highlight objects while dragging can be achieved with this trigger. When it happens, the mouse will have just crossed the boundary of an object while dragging some content with it.

See also:	`dataTransfer.dropEffect`, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDrag`, `onDragDrop`, `onDragEnd`, `onDragEnter`, `onDragLeave`, `onDragStart`, `onDrop`, Semantic event

onDragStart (Event handler)

The user has commenced some data selection with a mouse drag.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myObject`*`.ondragstart = `*`aHandler`*
HTML syntax:	`<HTMLTag onDragStart="...">`	
Argument list:	*`aHandler`*	A reference to a function object to handle the event.
Supported by objects:	`ACRONYM, DL, LI, STRONG, ADDRESS, Document, LISTING, SUB, B, DT, MARQUEE, SUP, BIG, EM, MENU, TABLE, BLOCKQUOTE, FIELDSET, OBJECT, TBODY, BODY, FONT, OL, TD, BUTTON, FORM, P, TEXTAREA, CAPTION, H1, PLAINTEXT, TFOOT, CENTER, HR, PRE, TH, CITE, I, Q, THEAD, CODE, IMG, S, TR, DD, Input, SAMP, TT, DEL, INS, Select, U, DFN, KBD, SMALL, UL, DIR, LABEL, SPAN, VAR, DIV, LEGEND, STRIKE,`	

At the commencement of a drag, this event trigger can be used to initiate some kind of feedback animation effect to tell the user where the dragged items can be deposited.

See also:	`dataTransfer.clearData()`, `dataTransfer.effectAllowed`, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDrag`, `onDragDrop`, `onDragEnd`, `onDragEnter`, `onDragLeave`, `onDragOver`, `onDrop`, Semantic event

onDrop (Event handler)

Called when a dragged object is dropped into a receiving window.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*`myObject`*`.ondrop = `*`aHandler`*

This is the complement of the `onDrag` event. It provides a means to hook into the object drop when it is deposited in the page.

See also:	`dataTransfer.clearData()`, `dataTransfer.dropEffect`, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onDrag`, `onDragDrop`, `onDragEnd`, `onDragEnter`, `onDragLeave`, `onDragOver`, `onDragStart`, Semantic event

onError (Event handler)

Triggered if an error occurs when loading an image.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onerror = *aHandler*
HTML syntax:	`<HTMLTag onError="..."><IMG onError="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`IMG, LINK, OBJECT, SCRIPT, STYLE`	

There is an `onError` event defined as part of the set of events supported by JavaScript. However, the `onError` event is actually only supported by `Image` objects when defined in HTML tag attributes. You can add an error handler to a `window` object with the `onerror` property.

The `<IMG onError="...">` tag attribute allows an error during image loading to be handled gracefully.

Attaching an error handler to a window with the `window.onerror` property allows JavaScript errors to be intercepted.

Error events have a slightly different parameter passing API so although they are events, they are slightly different from the rest of the event model.

The API for the function that is associated with this event should take three parameters.

Warnings:

- This event handler cannot be set in HTML using the tag attribute technique apart from when it applies to an `<IMG>` tag. However, it also applies to windows and frames although there is no tag attribute to connect it to. You can only associate it with a window or frame by using the script-driven property assignment.
- Beware of the return values. Returning `true` from an error handler inhibits the browser form carrying out any further action. This is exactly opposite to the return value from a form element event handler, which requires that a `false` value be returned to inhibit any further action by the browser.
- The range of objects you can register an `onError` handler with is platform and browser version dependent and you should experiment with your target client base if you plan to use this other than for debugging help.

See also:	Error events, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onAbort`, `onErrorUpdate`, `onLoad`, Semantic event, `Window.onerror`

Cross-references:

Wrox *Instant JavaScript* –page –55

onErrorUpdate (Event handler)

An error has occurred in the transfer of some data from a data source.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onerrorupdate = `*aHandler*
HTML syntax:	`<HTMLTag onErrorUpdate="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`APPLET, CAPTION, FIELDSET, RadioButton, AREA, Checkbox, Input, Select, BODY, Document, OBJECT, TEXTAREA`	

Data-driven systems may generate errors. This event is triggered when the error status of a database connection changes.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onError`, Semantic event

onFilterChange (Event handler)

A filter has changed the state of an element or a transition has just been completed.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onfilterchange = `*aHandler*
HTML syntax:	`<HTMLTag onFilterChange="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.

Supported by objects:	ACRONYM, DL, LISTING, STRONG, ADDRESS, DT, MENU, SUB, B, EM, OBJECT, SubmitButton, BIG, FIELDSET, OL, SUP, BLOCKQUOTE, FileUpload, P, TBODY, BODY, FONT, PLAINTEXT, TD, BUTTON, FORM, PRE, TEXTAREA, CAPTION, H1, Q, TFOOT, CENTER, HR, RadioButton, TH, Checkbox, I, ResetButton, THEAD,CITE, IMG, S, TR, CODE, Input, SAMP, TT, DD, INS, Select, U, DEL, KBD, SMALL, UL, DFN, LABEL, SPAN, VAR, DIR, LI, STRIKE,

Changing a filter will trigger this event. It is also triggered when a transition has just been completed. Filters are a MSIE-specific means of providing more attractive on-screen effects when things change.

Calling a function that changes a filter and executes a transition results in a further trigger for this event handler. Thus, you can set up an animation loop with this handler but, be careful to avoid stacking recursions or the browser will crash.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, `Filter` object, Handler, Semantic event, `style.filter`

onFinish (Event handler)

Triggered when a marquee object has finished looping.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onfinish = `*aHandler*
HTML syntax:	`<HTMLTag onFinish="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`MARQUEE`	

When the `MARQUEE` has completed its anticipated number of loops, this event is triggered so that you can set up some new animation if necessary.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onStart, onStop`, Semantic event

onFocus (Event handler)

When the form element is selected for entry.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –3.0 Netscape –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onfocus = *aHandler*
HTML syntax:	`<BODY onFocus="`*aHandler*`"> <FRAMESET onFocus="`*aHandler*`"> <HTMLTag onFocus="`*aHandler*`"> <INPUT onFocus="`*aHandler*`">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`A, DIV, LEGEND, TABLE, Anchor, Embed, MARQUEE, TEXTAREA, APPLET, FIELDSET, OBJECT, TextCell, AREA, FileUpload, Password, Url, BUTTON, IMG, RadioButton, CAPTION, Input, ResetButton, Checkbox, Layer, SubmitButton,`	

When the input focus is directed onto a control, the browser will generally highlight the item to indicate that the cursor is localized there and that any keyboard input will affect that `<INPUT>` item.

DOM level 2 events refers to this as a `DOMFocusIn` event, which employs event bubbling for its propagation and cannot be canceled.

In MSIE, you can hide the visible effect of an object acquiring focus by means of the `hideFocus` property on HTML elements.

Warnings:

- It is somewhat easy to create an `onBlur/onFocus` recursion, which leads to an endless loop with objects exchanging focus backwards and forwards. Focus on *`object A`* triggers a `blur` event on *`object B`*, which tries to wrest back the focus again because it believes its content is incomplete. Meanwhile *`object A`* isn't happy with its content either and so they play 'Tug-of-War' with one another. This continues until the browser has a race hazard attack and GPFs –that is, if it hasn't already fallen over due to a stack overflow. If you're lucky, only the browser will crash, but it could bring down the OS as well.

See also:	`Element.hideFocus`, Event, Event handler, Event model, Event names, Event object, `Event.returnValue`, Handler, `Input.blur(), Input.focus(), onBlur`, Password object, Semantic event, `UIEvent` object, `Window.onfocus`

onForward (Event handler)

When the forward button is clicked.

Availability:	JavaScript –1.3 Netscape –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myObject*.onforward = *aHandler*
HTML syntax:	`<HTMLTag onForward="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event

This event is called when the user interacts with the browser. Some events like this are helpful for blocking the operation of certain browser `UI` elements.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onBack`, Semantic event, Window events

onHelp (Event handler)

The user has pressed the `[F1]` key or selected `[help]` from the toolbar or menu.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*.onhelp = *aHandler*
HTML syntax:	`<BODY onHelp="..."> <FRAMESET onHelp="...">` `<HTMLTag onHelp="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`A, DIR, LI, STRIKE, ACRONYM, DIV, LISTING, STRONG, ADDRESS, DL, MAP, SUB, Anchor, Document, MARQUEE, SubmitButton, APPLET, DT, MENU, SUP, AREA, EM, OBJECT, TABLE, B, FIELDSET, OL, TBODY, BIG, FileUpload, P, TD, BLOCKQUOTE, FONT, Password, TEXTAREA, BODY, FORM, PLAINTEXT, TextCell, BUTTON, H1, PRE, TFOOT, CAPTION, HR, Q, TH, CENTER, I, RadioButton, THEAD, Checkbox, IMG, ResetButton, TR, CITE, Input, S, TT, CODE, INS, SAMP, U, DD, KBD, Select, UL, DEL, LABEL, SMALL, Url, DFN, LEGEND, SPAN, VAR`	

This handler is invoked if the `[F1]` key is pressed or the `[help]` item is selected from the menu while the `Element` object has the input focus.

This handler is particularly applicable to the <BODY> and <FRAMESET> tags and will be defined for Element objects that correspond to them.

In MSIE, after this event has been handled, the built-in help window will be displayed. This can be suppressed by passing the value false back rather than true. Passing no value back defaults to true.

Warnings:

- On the Macintosh platform, this event is triggered by the [HELP] key and not the [F1] key.

See also:	Element object, Event, Event handler, Event model, Event names, Event object, Event.returnValue, Handler, Semantic event

onKeyDown (Event handler)

Triggered when a key is pressed.

Availability:	HTML version –4.0 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera browser –3.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	-	*myObject*.onkeydown = *aHandler*
HTML syntax:	<A onKeyDown="..."><AREA onKeyDown="..."><BODY onKeyDown="..."><HTMLTag onKeyDown="..."><IMG onKeyDown="..."><INPUT onKeyDown="...">	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	A, DIR, LI, STRONG, ACRONYM, DIV, LISTING, SUB, ADDRESS, DL, MAP, SubmitButton, Anchor, Document, MARQUEE, SUP, APPLET, DT, MENU, TABLE, AREA, EM, OL, TBODY, B, FIELDSET, P, TD,BIG, FileUpload, Password, TEXTAREA, BLOCKQUOTE, FONT, PLAINTEXT, TextCell, BODY, FORM, PRE, TFOOT, BUTTON, H1, Q, TH, CAPTION, HR, RadioButton, THEAD, CENTER, I, ResetButton, TR, Checkbox, IMG, S, TT, CITE, Input, SAMP, U, CODE, INS, Select, UL, DD, KBD, SMALL, Url, DEL, LABEL, SPAN, VAR, DFN, LEGEND, STRIKE,	

As the key is pressed down, this event is triggered. If you return the value true, the event will be passed to the browser for further processing. Returning false inhibits this action and discards the message.

In Netscape, the key code of the key that triggered the event is available in the which property of the event object that is passed to the event handler as an argument. The equivalent value in MSIE is available in the keyCode property of the event object referenced by the window.event property.

The key value is the Unicode character code point value which, is a numeric value. This may need to be converted to a character if you want to display it. You can do that with the `String.fromCharCode()` static method which will create a one character string for you.

You may need to determine whether any of the modifier keys were pressed. In Netscape, you retrieve a bitmask value from the `modifiers` property of the `event` object. This can then be masked against the various bit flags available as static properties of the `Event` class.

In MSIE, modifier keys are available with a set of properties, which return the state of each modifier key individually.

In MSIE, you can modify the value of the key that is passed back to the browser by storing a different Unicode code point value in the `returnValue` property of the `Event` object.

Warnings:

- To use key-codes and modifier keys in a portable way, you will need to implement some browser-specific support and then call an appropriate routine according to the browser the script is executing in.
- This is not supported on the WebTV platform.

See also:	`Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.altKey, Event.ctrlKey, Event.keyCode, Event.modifiers, Event.returnValue, Event.shiftKey, Event.which`, Handler, JellyScript, Keyboard events, `onKeyPress, onKeyUp`, Semantic event, `String.fromCharCode(),TEXTAREA` object, `TextCell` object

onKeyPress (Event handler)

Pressing the key down and releasing it again triggers this event.

Availability:	HTML version –4.0 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onkeypress = *aHandler*
HTML syntax:	`<A onKeyPress="..."> <AREA onKeyPress="..."> <BODY onKeyPress="..."> <HTMLTag onKeyPress="..."> <IMG onKeyPress="..."> <INPUT onKeyPress="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event

Supported by objects:	A, DIR, LI, STRONG, ACRONYM, DIV, LISTING, SUB, ADDRESS, DL, MAP, SubmitButton, Anchor, Document, MARQUEE, SUP, APPLET, DT, MENU, TABLE, AREA, EM, OL, TBODY, B, FIELDSET, P, TD, BIG, FileUpload, Password, TEXTAREA, BLOCKQUOTE, FONT, PLAINTEXT, TextCell, BODY, FORM, PRE, TFOOT, BUTTON, H1, Q, TH, CAPTION, HR, RadioButton, THEAD, CENTER, I, ResetButton, TR, Checkbox, IMG, S, TT, CITE, Input, SAMP, U, CODE, INS, Select, UL, DD, KBD, SMALL, Url, DEL, LABEL, SPAN, VAR, DFN, LEGEND, STRIKE,

This handler is sometimes more appropriate than the `onKeyDown` and `onKeyUp` handlers. They should be used if you want some action to occur on each transition and to indicate by some feedback mechanism to the user that the key is still down.

As the key is pressed down and then released again, this event is triggered. If you return the value `true`, the event will be passed to the browser for further processing. Returning `false` inhibits this action and discards the message.

In Netscape, the key code of the key that triggered the event is available in the `which` property of the `event` object that is passed to the event handler as an argument. The equivalent value in MSIE is available in the `keyCode` property of the event object referenced by the `window.event` property.

The key value is the Unicode character code point value, which is a numeric value. This may need to be converted to a character if you want to display it. You can do that with the `String.fromCharCode()` static method, which will create a one character string for you.

You may need to determine whether any of the modifier keys were pressed. In Netscape, you retrieve a bitmask value from the `modifiers` property of the `event` object. This can then be masked against the various bit flags available as static properties of the `Event` class.

In MSIE, modifier keys are available with a set of properties that return the state of each modifier key individually.

In MSIE, you can modify the value of the key that is passed back to the browser by storing a different Unicode code point value in the `returnValue` property of the `Event` object.

This event was only supported by `TextCell` objects in the WebTV platform from the Summer 2000 release onwards.

Warnings:

- An `onKeyPress` event will also trigger an `onKeyDown` event and an `onKeyUp` event. The `onKeyPress` will not be triggered until the `onKeyUp` occurs. There is no guarantee that the `onKeyUp` will happen before the `onKeyPress` although that is the logical sequence.
- You do not receive a message when a modifier key is held down on its own. However you could sense the state of that as keys are processed. It isn't ideal but may be sufficient.

See also:	`Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, JellyScript, Keyboard events, `onKeyDown`, `onKeyUp`, Semantic event, `TEXTAREA` object, `TextCell` object

onKeyUp (Event handler)

Triggered when a key is released.

Availability:	HTML version –4.0 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera browser –3.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *myObject*.onkeyup = *aHandler*

HTML syntax:	`<AREA onKeyUp="..."> <BODY onKeyUp="...">`
	`<HTMLTag onKeyUp="..."> <IMG onKeyUp="...">`
	`<INPUT onKeyUp="...">`

Argument list:	*aHandler* A reference to a function object to handle the event
Supported by objects:	`A, DIR, LI, STRONG, ACRONYM, DIV, LISTING, SUB, ADDRESS, DL, MAP, SubmitButton, Anchor, Document, MARQUEE, SUP, APPLET, DT, MENU, TABLE, AREA, EM, OL, TBODY, B, FIELDSET, P, TD, BIG, FileUpload, Password, TEXTAREA, BLOCKQUOTE, FONT, PLAINTEXT, TextCell, BODY, FORM, PRE, TFOOT, BUTTON, H1, Q, TH, CAPTION, HR, RadioButton, THEAD, CENTER, I, ResetButton, TR, Checkbox, IMG, S, TT, CITE, Input, SAMP, U, CODE, INS, Select, UL, DD, KBD, SMALL, Url, DEL, LABEL, SPAN, VAR, DFN, LEGEND, STRIKE,`

As the key is released, this event is triggered.

In Netscape, the key code of the key that triggered the event is available in the `which` property of the `event` object that is passed to the event handler as an argument. The equivalent value in MSIE is available in the `keyCode` property of the event object referenced by the `window.event` property.

The key value is the Unicode character code point value, which is a numeric value. This may need to be converted to a character if you want to display it. You can do that with the `String.fromCharCode()` static method, which will create a one character string for you.

You may need to determine whether any of the modifier keys were pressed. In Netscape, you retrieve a bitmask value from the `modifiers` property of the `event` object. This can then be masked against the various bit flags available as static properties of the Event class.

In MSIE, modifier keys are available with a set of properties that return the state of each modifier key individually.

In MSIE, you can modify the value of the key that is passed back to the browser by storing a different Unicode code point value in the `returnValue` property of the `Event` object.

Warnings:

- This is not supported on the WebTV platform.

See also:	`Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, JellyScript, Keyboard events, `onKeyDown`, `onKeyPress`, Semantic event, `TEXTAREA` object, `TextCell` object

onLoad (Event handler)

Triggered when an object has completed loading.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onload = *aHandler*
HTML syntax:	`<BODY onLoad="aHandler"> <BODY onUnload="aHandler"> <FRAMESET onLoad="aHandler"> <FRAMESET onUnload="aHandler"> <HTMLTag onUnload="aHandler"> <IMG onUnload="aHandler">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`APPLET, BODY, IMG, LINK, STYLE, AREA, FRAMESET, Layer, SCRIPT, Window`	

This event is associated with `<BODY>` and `<FRAMESET>` tags. It is triggered when either have finished loading.

In the case of a `<BODY>` tag, that is when all images are loaded and any `<APPLET>` or plugin items have started running.

In the case of the `<FRAMESET>` tag, it is when all frames have loaded and arguably should be after all `<BODY>` tags have triggered their `onLoad` events. However, you probably should not rely on the last `<BODY>` `onLoad` event occurring before the `<FRAMESET>` `onLoad` event.

In any case, it would probably be bad form to have too many `onLoad` handlers activated during a page load.

Although it is associated with `<BODY>` and `<FRAMESET>`, the event is owned by the `window` object in the document object model.

Warnings:

- This fails to trigger on images when they are loaded into version 4.0 of MSIE. If you need an `onLoad` trigger for the page then you can trigger from the `<BODY onLoad="...">` HTML tag attribute. At least when the whole page is loaded your image should have loaded. However, even that isn't always reliable.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onAbort, onError, onUnload`, Semantic event, Window events, `Window.onload`, `Window.onunload`

onLoseCapture (Event handler)

Called when capturing is deactivated for an object.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myObject*.onlosecapture = *aHandler*

External event capture control may leave an object in some strange state. You may want to know when event capturing has been removed. This event is triggered when that happens.

See also:	`captureEvents`(), Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Semantic event

onMouseDown (Event handler)

Triggered when the mouse button is pressed.

Availability:	HTML version –4.0 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onmousedown = *aHandler*
HTML syntax:	`<A onMouseDown="..."> <AREA onMouseDown="..."> <BODY onMouseDown="..."><HTMLTag onMouseDown="..."> <IMG onMouseDown="..."> <INPUT onMouseDown="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.

Supported by objects:	A, DIR, LI, STRONG, ACRONYM, DIV, LISTING, SUB, ADDRESS, DL, MAP, SubmitButton, Anchor, Document, MARQUEE, SUP, APPLET, DT, MENU, TABLE, AREA, EM, OL, TBODY, B, FIELDSET, P, TD, BIG, FileUpload, Password, TEXTAREA, BLOCKQUOTE, FONT, PLAINTEXT, TextCell, BODY, FORM, PRE, TFOOT, BUTTON, H1, Q, TH, CAPTION, HR, RadioButton, THEAD, CENTER, I, ResetButton, TR, Checkbox, IMG, S, TT, CITE, Input, SAMP, U,CODE, INS, Select, UL, DD, KBD, SMALL, Url, DEL, LABEL, SPAN, VAR, DFN, LEGEND, STRIKE,

As the mouse button is pressed down, this event is triggered. If you return the value `true`, the event will be passed to the browser for further processing. Returning `false` inhibits this action and discards the message.

Netscape indicates which mouse button was pressed in the `which` property of the `event` object that is passed as an argument to the event handler function. MSIE makes the value available in the `button` property of the event object referenced by the `window.event` property.

DOM level 2 classifies this as a `MouseEvent` and specifies that it uses bubbling propagation and can be canceled.

Warnings:

- This is not supported on the WebTV platform.

See also:	`Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.button`, `Event.returnValue`, `Event.which`, Handler, JellyScript, Mouse events, `MouseEvent` object, `onClick`, `onDblClick`, `onMouseUp`, `onReset`, `onSelect`, `onSubmit`, Semantic event

onMouseDrag (Event handler)

An event handler for mouse drag operations.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onmousedrag = `*aHandler*
HTML syntax:	`<HTMLTag onMouseDrag="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event

Holding down the mouse button and then moving the mouse across the window will initiate this event.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Mouse events, Semantic event

onMouseMove (Event handler)

Triggered when the mouse pointer is moved.

Availability:	HTML version –4.0 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *myObject*.onmousemove = *aHandler*
HTML syntax:	`<HTMLTag onMouseMove="...">`
Argument list:	*aHandler* A reference to a function object to handle the event.
Supported by objects:	`A, DIR, LI, STRONG, ACRONYM, DIV, LISTING, SUB, ADDRESS, DL, MAP, SubmitButton, Anchor, Document, MARQUEE, SUP, APPLET, DT, MENU, TABLE, AREA, EM, OL, TBODY, B, FIELDSET, P, TD, BIG, FileUpload, Password, TEXTAREA, BLOCKQUOTE, FONT, PLAINTEXT, TextCell, BODY, FORM, PRE, TFOOT, BUTTON, H1, Q, TH, CAPTION, HR, RadioButton, THEAD, CENTER, I, ResetButton, TR, Checkbox, IMG, S, TT, CITE, Input, SAMP, U, CODE, INS, Select, UL, DD, KBD, SMALL, Url, DEL, LABEL, SPAN, VAR, DFN, LEGEND, STRIKE,`

When you move the mouse, the browser generates mouse move events and sends them to the `Element` object that is under the mouse at that moment. As the mouse reaches the boundary of an element a mouse event out and a mouse over event are generated and directed to the appropriate elements for them to handle.

Netscape does not support this event handler on individual elements. You have to use the `captureEvents()` method that belongs to a window or layer object and then delegate the handling to the handler as you need to.

DOM level 2 classifies this as a `MouseEvent` and specifies that it uses bubbling propagation and can be canceled.

Warnings:

- As a mouse is moved within an element, a large number of mouse move events will be triggered. This can cause problems if your handler carries out a lot of lengthy computation. It can cause severe browser crashing problems if you are creating and destroying objects under the mouse while it is moving.

See also:	`captureEvents()`, `Document.captureEvents()`, `Document.releaseEvents()`, `Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `Layer.captureEvents()`, `Layer.releaseEvents()`, Mouse events, `MouseEvent` object, Semantic event, `Window.captureEvents()`, `Window.releaseEvents()`

onMouseOut (Event handler)

Triggered when the mouse pointer leaves the active area occupied by the `Element` object that represents the object on screen.

Availability:	HTML version –4.0 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onmouseout = *aHandler*
HTML syntax:	`<A onMouseOut="..."> <AREA onMouseOut="..."> <HTMLTag onMouseOut="..."><IMG onMouseOut="..."> <LAYER onMouseOut="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`A, DIR, LEGEND, STRIKE, ACRONYM, DIV, LI, STRONG, ADDRESS, DL, LISTING, SUB, Anchor, Document, MAP, SubmitButton, APPLET, DT, MARQUEE, SUP,AREA, EM, MENU, TABLE, B, FIELDSET, OL, TBODY, BIG, FileUpload, P, TD, BLOCKQUOTE, FONT, Password, TEXTAREA, BODY, FORM, PLAINTEXT, TextCell, BUTTON, H1, PRE, TFOOT, CAPTION, HR, Q, TH, CENTER, I, RadioButton, THEAD, Checkbox, IMG, ResetButton, TR, CITE, Input, S, TT, CODE, INS, SAMP, U,DD, KBD, Select, UL, DEL, LABEL, SMALL, Url, DFN, Layer, SPAN, VAR`	

When you move the mouse, the browser generates mouse move events and sends them to the `Element` object that is under the mouse at that moment. As the mouse reaches the boundary of an element a mouse out event and a mouse over event are generated and directed to the appropriate elements for them to handle.

DOM level 2 classifies this as a `MouseEvent` and specifies that it uses bubbling propagation and can be canceled.

Warnings:

- You may need to implement some code for this handler to reset the status line after changing it with an `onMouseOver` event handler. That would be necessary for Netscape 2.0 and 3.0 on the Windows platform.
- As a mouse is moved over an element, an entry event is triggered. It can cause severe browser crashing problems if you are creating and destroying objects under the mouse while it is moving. In particular, the destruction of an object can mean that the mouse out has no logical destination since it must somehow leave the element it entered.

See also:	`Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Mouse events, `MouseEvent` object, `onMouseOver`, Semantic event, Status line, `Window.defaultStatus`, `Window.status`

onMouseOver (Event handler)

Triggered when the mouse pointer enters the active area owned by the object.

Availability:	HTML version –4.0 JavaScript –1.0 JScript –1.0 Internet Explorer –3.0 Netscape –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*`.onmouseover = `*aHandler*
HTML syntax:	`<A onMouseOver="..."> <AREA onMouseOver="..."> <HTMLTag onMouseOver="..."> <IMG onMouseOver="..."> <LAYER onMouseOver="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`A, DIR, LEGEND, STRIKE, ACRONYM, DIV, LI, STRONG, ADDRESS, DL, LISTING, SUB, Anchor, Document, MAP, SubmitButton, APPLET, DT, MARQUEE, SUP, AREA, EM, MENU, TABLE, B, FIELDSET, OL, TBODY, BIG, FileUpload, P, TD, BLOCKQUOTE, FONT, Password, TEXTAREA, BODY, FORM, PLAINTEXT, TextCell, BUTTON, H1, PRE, TFOOT, CAPTION, HR, Q, TH, CENTER, I, RadioButton, THEAD, Checkbox, IMG, ResetButton, TR, CITE, Input, S, TT, CODE, INS, SAMP, U, DD, KBD, Select, UL, DEL, LABEL, SMALL, Url, DFN, Layer, SPAN, VAR`	

As the mouse moves over the object, this event is triggered. Normally, passing over a link will display its value in the status bar. You can inhibit this behavior by returning the Boolean `true` value leaving the status line unchanged. Returning `false` allows the browser to overwrite the status line with the URL value of the link.

DOM level 2 classifies this as a `MouseEvent` and specifies that it uses bubbling propagation and can be canceled.

Warnings:

- You may need to implement an `onMouseOut` handler to clear the status line again if you used this event handler to display a message in the status line. That would be necessary in particular for Netscape 2.0 and 3.0 on the Windows platform.
- Be sure to set the Boolean return value correctly to either allow or deny the browser permission to perform its normal default behavior on return from the event handler.
- As a mouse is moved over an element, an entry event is triggered. It can cause severe browser crashing problems if you are creating and destroying objects under the mouse while it is moving. In particular, the destruction of an object can mean that the mouse out has no logical destination since it must somehow leave the element it entered.

See also:	`Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Mouse events, `MouseEvent` object, `onMouseOut`, Semantic event, Status line, `Window.defaultStatus, Window.status`

onMouseUp (Event handler)

Triggered when the mouse button is released.

Availability:	HTML version –4.0 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera browser –3.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *myObject*.onmouseup = *aHandler*
HTML syntax:	`<A onMouseUp="..."> <AREA onMouseUp="..."> <BODY onMouseUp="..."> <HTMLTag onMouseUp="..."> <IMG onMouseUp="..."> <INPUT onMouseUp="...">`
Argument list:	*aHandler* A reference to a function object to handle the event.
Supported by objects:	`A, DIR, LEGEND, STRONG, ACRONYM, DIV, LI, SUB, ADDRESS, DL, LISTING, SubmitButton, Anchor, Document, MAP, SUP, APPLET, DT, MARQUEE, TABLE, AREA, EM, MENU, TBODY, B, FIELDSET, OL, TD,BIG, FileUpload, P, TEXTAREA, BLOCKQUOTE, FONT, Password, TextCell, BODY, FORM, PLAINTEXT, TFOOT, BUTTON, H1, PRE, TH, CAPTION, HR, Q, THEAD, CENTER, I, ResetButton, TR, Checkbox, IMG, S, TT, CITE, Input, SAMP, U,CODE, INS, Select, UL, DD, KBD, SMALL, Url, DEL, LABEL, SPAN, VAR, DFN, Layer, STRIKE,`

As the mouse button is released, this event is triggered. If you return the value `true`, the event will be passed to the browser for further processing. Returning `false` inhibits this action and discards the message.

Netscape indicates which mouse button was pressed in the `which` property of the `event` object that is passed as an argument to the event handler function. MSIE makes the value available in the `button` property of the event object referenced by the `window.event` property.

DOM level 2 classifies this as a `MouseEvent` and specifies that it uses bubbling propagation and can be canceled.

Warnings:

- When you release the mouse button, you get a `MouseUp` raw event and a `Click` semantic event. Be careful that you know you are only calling event handlers once.
- This is not supported on the WebTV platform.

See also:	`Element` object, Event, Event handler, Event model, Event names, `Event` object, `Event.button, Event.returnValue, Event.which`, Handler, JellyScript, Mouse events, `MouseEvent` object, `onClick, onDblClick, onMouseDown, onReset, onSelect, onSubmit`, Semantic event

onMove (Event handler)

The browser window has been moved.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*`.onmove = `*aHandler*
HTML syntax:	`<BODY onMove="`*aHandler*`"> <FRAMESET onMove="`*aHandler*`"> <HTMLTag onMove="`*aHandler*`">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`Window`	

Moving the browser window usually won't cause any problems. However, if you have some window content that you want to position relative to the screen, or if you want to locate the window at a specific location, perhaps locating it on a grid, this event may be useful.

The event is triggered when a window is moved on the screen to another location.

The handler is registered by defining it with an HTML tag attribute or by assigning a handler function to the property.

The event is also triggered when a window is moved under control of a script with the `moveTo()` or `moveBy()` methods.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Semantic event, Window events, `Window.onmove`

onPaste (Event handler)

Called when a Paste operation is requested by the user.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*myObject*`.onpaste = `*aHandler*

This is useful if you want to intercept and possibly block a paste operation where the user may be attempting to paste some external content into a text field. Perhaps you do want to let them do it but you want to be sure the contents are checked for data integrity.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onCopy, onCut`, Semantic event

onPropertyChange (Event handler)

Called when a property belonging to the receiving object is changed.

Availability:	JScript –5.0 Internet Explorer –5.0	
JavaScript syntax:	IE	*myObject*.onpropertychange = *aHandler*

This is part of the property monitoring facilities. It can be used as an alternative to the `watch()/unwatch()` facilities in Netscape.

When this event is triggered, the handler can inspect the `event` object to determine what property was changed and on what object.

The object affected is returned in the `srcElement` property and the property in its `propertyName` property.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.propertyName`, `Event.returnValue`, `Event.srcElement`, Handler, Semantic event, `unwatch()`, `watch()`

onReadyStateChange (Event handler)

An object in the window has changed its ready state.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*.onreadystatechange = *aHandler*
HTML syntax:	`<HTMLTag onReadyStateChange="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`APPLET, Document, OBJECT, STYLE, AREA, LINK, SCRIPT,`	

This event fires when the state of the object has changed.

Some objects may need to be completely loaded before you can properly interact with them. This is a means of getting a call-back when the ready state of an object changes. It saves the need for watching the `readyState` in a polling loop, which is wasteful of resources and a generally deprecated technique.

See also:	`Document.readyState`, `Element.readyState`, `Embed.readyState`, Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `IMG.readyState`, `LINK.readyState`, `OBJECT.readyState`, `SCRIPT.readyState`, Semantic event, `STYLE.readyState`, `XML.event`

onReset (Event handler)

The user has clicked a reset button in a form.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onreset = *aHandler*
HTML syntax:	`<FORM onReset="..."> <HTMLTag onReset="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`FORM`	

As the `<FORM>` reset button is clicked, this event is triggered. If you return the value `true`, the event will be passed to the browser for further processing and the form will be reset. Returning `false` inhibits this action and discards the message, and the form will not be reset. This provides a means of inhibiting `<FORM>` resets altogether.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onClick, onMouseDown, onMouseUp, onSubmit`, Semantic event

onResize (Event handler)

As the window is resized, this event is triggered.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onresize = *aHandler*
HTML syntax:	`<BODY onResize="`*aHandler*`"> <FRAMESET onResize="`*aHandler*`"> <HTMLTag onResize="`*aHandler*`">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`APPLET, FIELDSET, MARQUEE, Input, AREA, FRAMESET, Select, TD, BUTTON, IMG, TABLE, Window, DIV, , ,`	

Moving or resizing windows may be something your scripts will need to know about. A move is generally harmless, but a resize may not be so benign to your page content. This event trigger provides a means to fix up the display if the window aspect ration or size is changed.

This event is triggered when a window is enlarged or reduced in size.

The handler is registered by defining it with an HTML tag attribute. The handler can also be registered by assigning the `function` object to the `onresize` property of the window.

The event is also triggered when a window is resized under control of a script with the `resizeTo()` or `moveBy()` methods.

Warnings:

- There are numerous issues with `onResize` events in Netscape 4.0. In particular, an `onResize` causes Netscape to "forget" positioning information for `<LAYER>` objects and absolutely positioned `<DIV>` blocks.
- This could be fixed by simply making this assignment:

```
self.onresize = self.reload;
```

- This attaches a reload call to the resize event handler. However registering event handlers in this way on Netscape prior to version 6.0 has proven to be inconsistent. In this particular case it will cause a browser-crashing endless loop.
- The crashing only happens on some versions of Netscape 4.0. The problem is caused because the `resize` event handler will fire a second `Resize` event when the window scrollbars are drawn once the page is loaded –odd behavior. We've sat in front these browsers for years now and watched Netscape draw a page, then realize it can't fit all the content in and so decide it needs scroll bars and draw it all again. However, up until recently we never realized that an `onResize` was being triggered as well –if only we had a generic event watcher of some kind so we could see what is going on behind the scenes!
- Anyway, the example suggests a fix and was provided by Jon Stephens. This demonstrates other subtle and neat tricks such as storing persistent values as member properties of the function object itself.

Example code:

```
// This checks to see if the window's dimensions have
// actually changed (because Netscape often fires a
// false onResize event when it forms scrollbars);
// if the dimensions have changed, the document is
// reloaded.
// Note that document.location is not supposed to be
// settable, but here's another case where the
// implementation does not match the specs.
function resizeFix()
{
   if((document.resizeFix.initWidth  != window.innerWidth)
||(document.resizeFix.initHeight != window.innerHeight))
   {
      document.location = document.location;
   }
}

// This function checks to see if the browser supports the
// Layer object (i.e., Netscape 4.X);
// If it does, then it creates a new object with properties
// to hold the current window width & height and assigns
// the resizeFix() function to the window's onResize event.
function checkBrowser()
```

```
{
   if(document.layers)
   {   if(typeof document.fix == "undefined")
       {
           document.resizeFix = new Object();
           document.resizeFix.initWidth  = window.innerWidth;
document.resizeFix.initHeight = window.innerHeight;
       }
       window.onresize = resizeFix;
   }
}

// This calls the browser check function above
checkBrowser();
```

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, Semantic event, Window events, `Window.onresize`

onRowEnter (Event handler)

The data in a field bound to a data source is about to be changed.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onrowenter = `*aHandler*
HTML syntax:	`<HTMLTag onRowEnter="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`APPLET, DIV, MARQUEE, TD, AREA, Document, OBJECT, TEXTAREA, BODY, IMG, Select, TH, BUTTON, Input, TABLE,`	

This event is triggered for those data driven systems that need to track row and column changes to the data containers that are associated with a database. It fires to indicate that the current row has changed in the data source and new data values are available on the object.

See also:	Event, Event handler, Event model, Event names, Event object, `Event.returnValue`, Handler, `onRowExit`, Semantic event, `XML` object, `XML.event`

onRowExit (Event handler)

The data in a field bound to a data source has been changed.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onrowexit = `*aHandler*
HTML syntax:	`<HTMLTag onRowExit="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`APPLET, DIV, MARQUEE, TD, AREA, Document, OBJECT, TEXTAREA, BODY, IMG, Select, TH, BUTTON, Input, TABLE,`	

This event fires just before the data source control changes the current row in the object.

On exiting from a row, you may want to check the contents and possibly update the database with the changes.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onRowEnter`, Semantic event, `XML` object, `XML.event`

onRowsDelete (Event handler)

Some rows are about to be deleted from the database.

Availability:	JScript –3.0 Internet Explorer –4.0
Property/method value type:	`Boolean primitive`

This event fires when rows are about to be deleted from the recordset. You may want to intercept this event and check whether you really do want to delete these rows.

See also:	`XML.event`

onRowsInserted (Event handler)

Some new data is being inserted into the database.

Availability:	JScript –3.0 Internet Explorer –4.0
Property/method value type:	`Boolean primitive`

This event fires just after new rows are inserted in the current recordset. You may want to intercept this event to fix up some relationships in the database to maintain data integrity.

See also:	XML.event

onScroll (Event handler)

The window has been scrolled.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	IE	*myObject*.onscroll = *aHandler*
HTML syntax:	<HTMLTag onScroll="...">	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	BODY, FIELDSET, MARQUEE, Window, CAPTION, IMG, TABLE, DIV, LEGEND, TEXTAREA,	

Scrolling windows is usually harmless. However, you might have some window content that want to remain visible after the window has scrolled. Perhaps you have some navigation objects placed in a layer that you want to leave positioned where it was regardless of the window scroll position.

See also:	Event, Event handler, Event model, Event names, Event object, Event.returnValue, Handler, Semantic event, Window events

onSelect (Event handler)

Some textual content in the window has been selected.

Availability:	JavaScript –1.0 JScript –3.0 Internet Explorer –4.0 Netscape –2.0 Opera browser –3.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	-	*myObject*.onselect = *aHandler*
HTML syntax:	<HTMLTag onSelect="...">	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	CAPTION, FileUpload, RadioButton, TEXTAREA, Checkbox, Input, ResetButton, TextCell, FIELDSET, Password, SubmitButton,	

Selecting text in the window may be a precursor to a cut or paste operation. If the elements are able to support TextRanges, then you may want to receive a trigger when a text selection occurs.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onClick, onMouseDown, onMouseUp, onSelectStart`, Semantic event

onSelectStart (Event handler)

A select action is beginning.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onselectstart = `*aHandler*
HTML syntax:	`<HTMLTag onSelectStart="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event.
Supported by objects:	`A, DIV, LI, SUB, ACRONYM, DL, LISTING, SUP, ADDRESS, Document, MARQUEE, TABLE, Anchor, DT, MENU, TBODY, B, EM, OBJECT, TD, BIG, FIELDSET, OL, TEXTAREA, BLOCKQUOTE, FONT, P, TFOOT, BODY, FORM, PLAINTEXT, TH, BUTTON, H1, PRE, THEAD, CAPTION, HR, Q, TR, CENTER, I, S, TT, CITE, IMG, SAMP, U, CODE, Input, Select, UL, DD, INS, SMALL, Url, DEL, KBD, SPAN, VAR, DFN, LABEL, STRIKE, , DIR, LEGEND, STRONG,`	

At the commencement of a selection, this event is triggered. This allows you to do some animation or perhaps draw a bounding rectangle while the selection takes place.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onSelect,` Semantic event

onStart (Event handler)

Fires when a `MARQUEE` element is beginning its loop.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myObject*`.onstart = `*aHandler*
HTML syntax:	`<HTMLTag onStart="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`MARQUEE`	

A `MARQUEE` element provides a suite of events that allow you to attach all manner of event driven animations and wizzy effects to them. This event is triggered when the `MARQUEE` starts its loop.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onBounce, onFinish, onStop`, Semantic event

onStop (Event handler)

Called when the user stops a page loading.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myObject*`.onstop = `*aHandler*

This event is called when the user halts the page loading with the stop button on the toolbar.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onBounce, onFinish, onStart`, Semantic event

onSubmit (Event handler)

The user has clicked on the submit button in a form.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –3.0 Netscape –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*`.onsubmit = `*aHandler*
HTML syntax:	`<FORM onSubmit="..."> <HTMLTag onSubmit="...">`	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`FORM`	

As the `<FORM>` submit button is clicked, this event is triggered. If you return the value `true`, the event will be passed to the browser for further processing and the form will be submitted. Returning `false` inhibits this action and discards the message, the form will not be submitted. This provides a means of inhibiting `<FORM>` submits when the form data is bad.

You can later on call the `submit()` method to send the form content back to the server.

Note that the `submit()` method will not trigger an `onSubmit` event.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, `onClick, onMouseDown, onMouseUp, onReset`, Semantic event

onUnload (Event handler)

Triggered when the document is unloaded.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myObject*.onunload = *aHandler*
HTML syntax:	<BODY onUnload="*aHandler*"> <FRAMESET onUnload="*aHandler*"> <HTMLTag onUnload="*aHandler*">	
Argument list:	*aHandler*	A reference to a function object to handle the event
Supported by objects:	`BODY, Layer, SCRIPT, Window, FRAMESET, LINK, STYLE,`	

The `onUnload` event is triggered for a `<BODY>` or `<FRAMESET>` tag when the page is about to disappear and be replaced with another. You should avoid doing anything lengthy or complex here –it should just be used to do some cleaning up.

It is considered bad technique to place an alert or any other dialog in the `onUnload` handler. It is also bad manners to tie the processing up in a lengthy piece of script execution at this stage.

You might use the `unLoad` handler to remove any values you put into the window status bar.

Although this is a tag attribute that is applied to the `<BODY>` tag, it is really associated with the `Window` object in which the document body is loaded.

Warnings:

- This is not supported on the WebTV platform.

See also:	Event, Event handler, Event model, Event names, `Event` object, `Event.returnValue`, Handler, JellyScript, `onBeforeUnload`, `onLoad`, Semantic event, Window events, `Window.onunload`

open() (Method)

An alias for the `window.open()` method.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Window` object	
JavaScript syntax:	-	*myWindow*.open()
	-	*myWindow*.open(*aURL*)
	-	*myWindow*.open(*aURL*, *aName*)
	-	*myWindow*.open(*aURL*, *aName*, *aFeatureList*)
	-	*myWindow*.open(*aURL*, *aName*, *aFeatureList*, *aFlag*)
	-	open()
	-	open(*aURL*)
	-	open(*aURL*, *aName*)
	-	open(*aURL*, *aName*, *aFeatureList*)
	-	open(aURL, *aName*, *aFeatureList*, *aFlag*)
Argument list:	*aFeatureList*	A list of attributes for the new window
	aFlag	A flag to indicate how the history list is to be modified
	aName	The name of a new or existing target window
	aURL	A URL to load into the window

Refer to:

`Window.open()`

opener (Property)

An alias for the `window.opener` property.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera browser –3.0	
Property/method value type:	`Window` object	
JavaScript syntax:	-	*myWindow*.opener
	-	opener

Property attributes:

`ReadOnly.`

Refer to:

`Window.opener`

OpenTV (TV Set-top Box)

An interactive TV set-top box environment.

At present this system does not support HTML or JavaScript. However, there is a growing trend among TV set-topbox manufacturers to provide integrated web browsing.

Because the developer environment for this system is based on the C language, it is theoretically possible to download a minimal HTML and JavaScript browser into the set-top box. There would be major considerations regarding bandwidth requirements for such an application but it could be done.

As the set-top boxes become more sophisticated, expect this kind of functionality to become commonplace. It becomes even more likely with the advent of large hard disk caches and more powerful CPUs driving the set-top box.

See also:	Interpret, Platform, Script execution, TV Set-top boxes, Web browser

Opera (Web browser)

A web browser alternative to MSIE and Netscape.

The Opera web browser is an alternative browser to the Netscape and MSIE browsers on the Windows platform. It is highly standards-based, and as a result has gained much respect. However, because many pages do not contain strictly standard content, you should quality-check your site on this browser to ensure its compliance. Using Opera as a reference browser can significantly improve the likelihood of your site continuing to work in the future as all browsers are likely to converge on the functionality embodied in the standards.

Details of what is supported in version 5 can be found at the web reference below.

See also:	ECMA, ECMAScript, Platform, Script execution, Web browser

Web-references:

http:www.opera.com/opera5/

Operator (Definition)

A special symbolic token that when placed adjacent to or between values creates an expression.

Availability:	ECMAScript edition –2

An operator is a token that represents a particular operation to be performed on one or two operands. The combination of operands and operator is called an expression. The operation yields a result, which can be used as an operand in subsequent expressions. Expressions can be nested implicitly according to the precedence rules of the operators, or explicitly by using using the parentheses grouping operator.

Depending on the operators and operands used, side effects may occur. This is particularly likely when function calls are used as operands since a function call can invoke many lines of code.

For example, an expression calling two functions, each of which opens a pop-up browser window would cause two new windows to appear as a side effect every time the expression was evaluated.

Here is a list of all the operators supported by JavaScript:

Operator	Description
!	Logical NOT
!=	NOT equal to
%	Remainder
%=	Remainder and assign to an LValue
&	Bitwise AND
&&	Logical AND
&=	Bitwise AND and assign to an LValue
(	Function argument delimiter and precedence control
)	Function argument delimiter and precedence control
*	Multiply
*=	Multiply and assign to an LValue
+	Add
+	Concatenate string
+	Unary convert the operand to a numeric value.
++	Increment LValue
+=	Add and assign to an LValue
,	Argument delimiter
-	Subtract
-	Unary negate the value
--	Decrement LValue

Table continued on following page

Operator	Description
`-=`	Subtract and assign to an LValue
`.`	Property accessor
`/`	Divide
`/=`	Divide and assign to an LValue
`:`	Part of conditional operator
`;`	Empty statement
`<`	Less than
`<<`	Bitwise left shift
`<<=`	Bitwise shift left and assign to an LValue
`<=`	Less than or equal to
`=`	Simple assignment to an LValue
`==`	Equal to
`===`	Identity
`>`	Greater than
`>=`	Greater than or equal to
`>>`	Bitwise shift right
`>>=`	Bitwise shift right and assign to an LValue
`>>>`	Bitwise shift right (unsigned)
`>>>=`	Bitwise shift right (unsigned) and assign to an LValue
`?`	Conditional operator
`[`	Array index delimiter
`]`	Array index delimiter
`^`	Bitwise XOR (exclusive OR)
`^=`	Bitwise exclusive XOR and assign to an LValue
`delete`	Used to delete a property from an object if it can be deleted
`false`	Boolean constant
`new`	Invokes an object constructor
`true`	Boolean constant
`typeof`	Determines the type of an evaluation or value
`void`	Regardless of the result of evaluating the expression that may be operated on, this will always yield the undefined value
`{`	Start code block
`|`	Bitwise inclusive OR
`|=`	Bitwise inclusive OR and assign to an LValue
`||`	Logical OR
`}`	End code block
`~`	Bitwise complement (NOT)

Refer to the Operator Precedence topic for details of the order in which operators and their expressions are evaluated in complex lines of code. This is sometimes light-heartedly referred to as "who's on first".

See also:	Additive operator, Assignment operator, Associativity, Binary operator, Bitwise operator, Cast operator, Equality operator, Expression, Grouping operator ( ), Lexical element, Logical operator, Mathematics, Multiplicative operator, Operator Precedence, Postfix operator, Prefix operator, Punctuator, `typeof`, Unary operator, `void`

Cross-references:

ECMA 262 edition 2 –section 11

ECMA 262 edition 3 –section 11

Operator Precedence (Definition)

A means of controlling execution priority in expressions.

Availability:	ECMAScript edition –2

The operators below are listed in decending order of precedence. The associativity colum indicates the direction in which items are evaluated.

Operator	Description	Assoc
`()`	Grouping operator	L-R
`[]`	Array index delimiter	L-R
`.`	Property accessor	L-R
`++`	Postfix increment	L-R
`--`	Postfix decrement	L-R
`!`	Logical NOT	R-L
`~`	Bitwise NOT	R-L
`++`	Prefix increment	R-L
`--`	Prefix decrement	R-L
`-`	Negate operand	L-R
`delete`	Delete a property from an object	R-L
`new`	Invokes an object constructor	R-L
`typeof`	Determines the type of a value	R-L
`void`	Always yields the undefined value	R-L
`*`	Multiply	L-R
`/`	Divide	L-R
`%`	Remainder	L-R
`+`	Convert the operand to a numeric value	L-R
`+`	Add	L-R
`-`	Subtract	L-R

Table continued on following page

Operator	Description	Assoc
`+`	Concatenate string	L-R
`<<`	Bitwise shift left	L-R
`>>`	Bitwise shift right	L-R
`>>>`	Bitwise shift right (unsigned)	L-R
`<`	Compare less than	L-R
`<=`	Compare less than or equal to	L-R
`>`	Compare greater than	L-R
`>=`	Compare greater than or equal to	L-R
`in`	Property is in object	L-R
`instanceof`	Object is instance of another object	L-R
`==`	Compare equal to	L-R
`!=`	Compare NOT equal to	L-R
`===`	Compare identically equal to	L-R
`!==`	Compare identically NOT equal to	L-R
`&`	Bitwise AND	L-R
`^`	Bitwise XOR	L-R
`|`	Bitwise OR	L-R
`&&`	Logical AND	L-R
`||`	Logical OR	L-R
`?:`	Conditional execution	R-L
`=`	Assign	R-L
`*=`	Multiply and assign	R-L
`/=`	Divide and assign	R-L
`%=`	Remainder and assign	R-L
`+=`	Add and assign	R-L
`-=`	Subtract and assign	R-L
`<<=`	Bitwise shift left and assign	R-L
`>>=`	Bitwise shift right and assign	R-L
`>>>=`	Bitwise shift right (unsigned) and assign	R-L
`&=`	Bitwise AND and assign	R-L
`|=`	Bitwise inclusive OR and assign	R-L
`^=`	Bitwise XOR and assign	R-L
`,`	Argument delimiter	L-R
`;`	Empty statement	L-R
`{ }`	Delimit code block	L-R

See also:	Associativity, Object property delimiter (.), Primary expression

Cross-references:

ECMA 262 edition 2 –section 11.1.4

ECMA 262 edition 3 –section 11.1.6

OptGroupElement object (Object/HTML)

A means of grouping options together into logical sets.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`myOPTGROUP = myDocument.all.anElementID`
	IE	`myOPTGROUP = myDocument.all.tags("OPTGROUP")[anIndex]`
	IE	`myOPTGROUP = myDocument.all[aName]`
	-	`myOPTGROUP = myDocument.getElementById(anElementID)`
	-	`myOPTGROUP = myDocument.getElementsByName(aName)[anIndex]`
	-	`myOPTGROUP = myDocument.getElementsByTagName("OPTGROUP")[anIndex]`
HTML syntax:	`<OPTGROUP> ... </OPTGROUP>`	
Argument list:	*anElementID*	The `ID` value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Object properties:	`disabled, label`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`disabled`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`label`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	5.0 +	-	5.0 +	-	-	-	Warning
onKeyDown	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	5.0 +	6.0 +	5.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

OptGroupElement.disabled (Property)

A switch for activating or deactivating a grouped option set.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`.myOptGroupElement`*`.disabled`

OptGroupElement.label (Property)

A label that is applied to a group of options.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myOptGroupElement`*`.label`

Option object (Object/HTML)

One of a set of objects belonging to a select object in a form.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera browser –3.0	
Inherits from:	`Element` object	
JavaScript syntax:	-	*myOption* = *myDocument*.*aFormName*.*aSelectorName*.options[*anIndex*]
	-	*myOption* = *myDocument*.*aFormName*.elements[*anItem*Index].options[*anIndex*]
	IE	*myOption* = *myDocument*.all.*anElementID*
	IE	*myOption* = *myDocument*.all.*anElementID*.elements[*anIndex*].options[*anIndex*]
	IE	*myOption* = *myDocument*.all.*anElementID*.options[*anIndex*]
	IE	*myOption* = *myDocument*.all.tags("OPTION")[*anIndex*]
	IE	*myOption* = *myDocument*.all[*aName*]
	-	*myOption* = *myDocument*.forms[*aFormIndex*].*aSelectorName*.options[*anIndex*]
	-	*myOption* = *myDocument*.forms[*aFormIndex*].elements[*anIndex*].options[*anIndex*]
	-	*myOption* = *myDocument*.getElementById(*anElementID*)
	-	*myOption* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myOption* = *myForm*.*aSelectorName*.options[*anIndex*]
	-	*myOption* = *myForm*.*elements*[*anItem*Index].options[*anIndex*]
	-	*myOption* = *myOptionsArray*[*anIndex*]
	-	*myOption* = *mySelector*.options[*anIndex*]
	-	*myOption* = *myDocument*.getElementsByTagName("OPTION")[*anIndex*]
HTML syntax:	`<OPTION> ... </OPTION>`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	anItemIndex	A valid reference to an item in the collection
	aName	The name attribute of an element
	aFormIndex	A reference to a particular form in the forms collection
	anElementID	The `ID` attribute of an element
Object properties:	`defaultSelected, form, index, label, prototype, selected, text, value`	

Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`

In Netscape, this sub-class of the Input object supports a couple of properties that may not be available on other platforms.

The DOM level 1 specification calls this object type an `OptionElement` object.

Warnings:

- In Netscape, this object is easy to confuse with the `Select` object to which the options belong. Be careful to maintain the correct structural relationship between `Select` popup menus and their option sets.
- Netscape 6.0 PR3 exhibited some instabilities in the support of this object; however, it is not certain whether the bug is still outstanding on the final release as well. The problem seemed related to the creation of new `Option` objects by means of the constructor. This is not something that everyone is going to be doing on all their pages so its not likely to be a show stopper unless its just that one thing you need to use.

See also:	`Form.elements[]`, Input object, `OptionsArray` object, `response.getOptionValue()`, `response.getOptionValueCount()`, `Select` object

Property	JavaScript	JScript	N	IE	Opera	NES	DOM	HTML	Notes
`defaultSelected`	1.1 +	1.0 +	3.0 +	3.02 +	3.0 +	2.0 +	1 +	-	-
`form`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	1 +	-	Warning
`index`	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	-	ReadOnly
`label`	1.5 +	-	6.0 +	-	-	-	1 +	-	-
`prototype`	1.0 +	1.0 +	2.0 +	3.02 +	-	2.0 +	-	-	Warning
`selected`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	1 +	-	-
`text`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	1 +	-	ReadOnly
`value`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	2.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	NES	DOM	HTML	Notes
`onClick`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	NES	DOM	HTML	Notes
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

Option() (Constructor)

An Option object constructor.

Availability:	JavaScript –1.1 Netscape –3.0	
JavaScript syntax:	N	`new Option`
	N	`new Option()`
	N	`new Option(`*`aValue`*`)`
Argument list:	*`aValue`*	An initial value for the new option object

You can dynamically create new `option` objects and add them to the `options` array belonging to a `Select` object in a form.

See also:	Constructor function, constructor property, Garbage collection, Memory leak, Reference counting, `Select.options[]`

Option.defaultSelected (Property)

The selected state of this item when the form was created.

Availability:	DOM level –1 JavaScript –1.1 JScript –1.0 Internet Explorer –3.02 Netscape –3.0 Netscape Enterprise Server version –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`myOption`*`.defaultSelected`

This is a Boolean value that indicates whether the `<OPTION>` tag in the HTML document source has the `SELECTED` HTML tag attribute present or not. Only one of the `<OPTION>` tags in a `<SELECT>` block should have the `SELECTED` attribute, unless multiple sections are allowed.

You can refer to this value in the form checking script to see if the value of the select block has been changed since the page was loaded.

Option.index (Property)

The index position within the select object set.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myOption*.index

When you define `<OPTION>` tags within a `<SELECT>` block, they instantiate objects to represent each option. These are then made available as members of a collection belonging to the `Select` object and accessible via its `options[]` property.

This property indicates the position of this `Option` object within that collection. The first `Option` object in the collection is located at index position zero. Note that modifying the set of objects in this collection can cause the index numbers of an option to change.

See also:	`Select.options[]`

Property attributes:

`ReadOnly`.

Option.label (Property)

The text string that the user sees in the popup menu. This is another name for the `Option.text` property.

Availability:	DOM level –1 JavaScript –1.5 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myOption*.label

Refer to:

`Option.text`

Option.prototype (Property)

The prototype for the `Option` object that can be used to extend the interface for all `Option` objects.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0	
Property/method value type:	`Option` object	
JavaScript syntax:	-	`Option.prototype`
	-	*`myOption`*`.constructor.prototype`

Warnings:

- Of all the objects supported by MSIE, this is the only one to provide an enumerable property that points at the prototype for the object. This is likely to be a bug in the enumeration flag settings for the `Option` object type.

See also:	`prototype` property

Option.selected (Property)

The selected state of this option item.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera browser –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`myOption`*`.selected`

This is the current selected state of the option item. If the user has not interacted with the parent `Select` object, this value will be the same as its initial `defaultSelected` value. If the user has chosen an alternative item from the collection, then this property will be set accordingly.

Option.text (Property)

The text string that the user sees in the popup menu.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera browser –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOption*.text

Option items have a textual value enclosed between the opening and closing tags. This is a means of accessing that text without needing to resort to complex `innerText` or `innerHTML` tricks.

Property attributes:

`ReadOnly.`

Option.value (Property)

The text value that is returned to the server if this item is selected.

Availability:	DOM level –1 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server version –2.0 Opera browser –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myOption*.value

The text that the user sees is not necessarily the same as that sent back when the form is submitted. Some degree of normalization is available so that the user may see a text string but a lookup mechanism can replace that with a numeric value thereby saving some work in the server.

See also:	`Input.value`

OptionElement object (Object/DOM)

One of a set of objects belonging to a select object in a form.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
JavaScript syntax:	-	*myOptionElement* = new OptionElement()

Refer to:

`Option` object

OptionsArray object (Object/browser)

A collection object that belongs to a select popup.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
JavaScript syntax:	-	*myOptionsArray* = *mySelector*.options
Object properties:	length	
Object methods:	add(), item(), remove(), select()	

Warnings:

- Netscape 6.0 implements this in a DOM-compliant manner. That means the object type is an `HTMLCollection` object. This is a generic object type and there are no special methods or properties added to it. The things you might have done with an `OptionsArray` object are not going to work.
- MSIE appears to implement this as a `NodeList` although you cannot tell because it doesn't make a constructor or prototype available for you to inspect.

See also:	`Collection` object, `Option` object, `Select` object, `Select.options[]`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
length	1.0 +	1.0 +	2.0 +	3.02 +	-	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
`add()`	-	3.0 +	-	4.0 +	-	-	-
`item()`	-	3.0 +	-	4.0 +	-	-	-
`remove()`	-	3.0 +	-	4.0 +	-	-	-
`select()`	-	3.0 +	-	4.0 +	-	-	-

OptionsArray.add() (Method)

Adds a new option item to a select popup object.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*myOptionsArray*.add(*anObject*)
	IE	*myOptionsArray*.add(*anObject*, *anIndex*)
Argument list:	*anObject*	An object to be added to the collection
	anIndex	The index position at which it should be added

You can use this method call to add a new item to a list of options belonging to a `Select` object. This would be manifested to the user as a new item being visible when the select menu is popped onto the display.

OptionsArray.item() (Method)

Access to a particular item in the options collection of a select popup object.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Option` object	
JavaScript syntax:	IE	*myOptionsArray*.item(*anIndex*)
	IE	*myOptionsArray*.item(*aSelector*)
	IE	*myOptionsArray*.item(*aSelector*, *anIndex*)
Argument list:	*anIndex*	A zero based index into the collection
	aSelector	A textual value that selects all matching objects

Because the `OptionsArray` is a sub-class of the generic `Collection` object, all the normal things you can do with a collection are possible with an `OptionsArray` belonging to a `Select` object.

See also:	`Collection.Item()`

OptionsArray.length (Property)

The number of options in a select popup.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myOptionsArray`*`.length`

Property attributes:

`ReadOnly.`

Refer to:

`Collection.length`

OptionsArray.remove() (Method)

Removes an option from the collection belonging to a `select` popup object.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*`myOptionsArray`*`.remove(`*`anIndex`*`)`
Argument list:	*`anIndex`*	A zero-based index into the collection

This should affect the display causing the popup at least to be refreshed. Manipulating the popup contents in this way may be useful and is particularly appropriate if you have two popups, one of which is a secondary selector whose available choices depend on the settings of the first.

OptionsArray.select() (Method)

Select an item in an options array belonging to a select popup object.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	*`myOptionsArray`*`.select()`

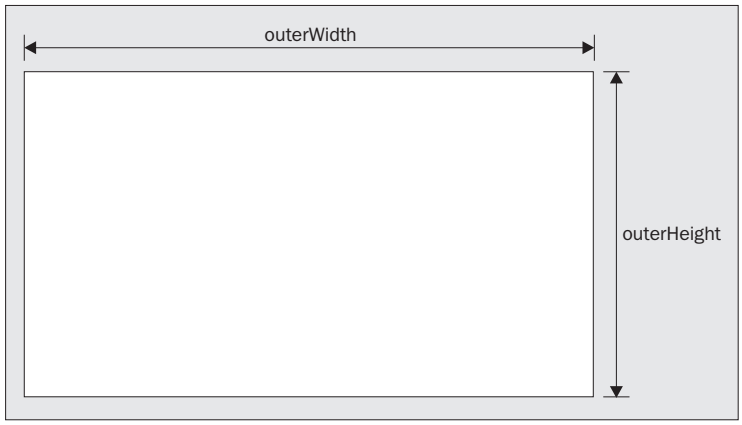
Behaves as if the user had selected an item. This should highlight the item in the popup's usual user interface feedback manner and should set the appropriate values in the properties of the previously selected and freshly selected `Option` objects.

See also:	`Input.select()`

outerHeight (Property)

An alias for the `window.outerHeight` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myWindow*`.outerHeight`
	-	`outerHeight`



Property attributes:

`ReadOnly.`

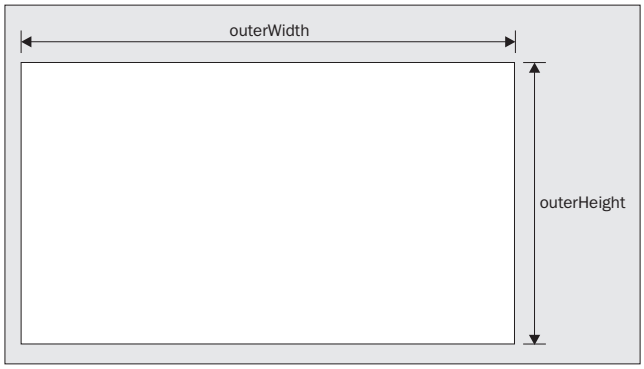
Refer to:

`Window.outerHeight`

outerWidth (Property)

An alias for the `window.outerWidth` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`myWindow.outerWidth`
	-	`outerWidth`



Property attributes:

`ReadOnly.`

Refer to:

`Window.outerWidth`

Overview (Background)

JavaScript language and functionality overview.

Availability:	ECMAScript edition –2

JavaScript originally started as a Netscape extension to provide some script-driven dynamic effects within the web page. At this time, Netscape had by far the greater penetration and was much more popular than the MSIE browser.

MSIE rapidly caught up with the scripting functionality and extended it in different directions. In the typically competitive style of the Microsoft company, JScript rapidly became the equal of the Netscape 4.0 browser. However, now it's the turn of Netscape to up the stakes again with the introduction of Netscape 6.0, although Microsoft has a new version of MSIE at the stage of beta testing.

In the two years Netscape has taken in releasing new browser, JavaScript has become standardized through the ECMA organization and has been deployed in a variety of non-browser contexts. It is now part of the fundamental scripting interface in Windows where its duty with the VBScript interpreter lies in automating desktop operations.

JavaScript is now available on Unix platforms as a shell scripting language and in web servers for server-side programming.

More recently, it has been adopted and modified to become WMLScript which is used in WAP-standard mobile phone. Even more recently it is becoming popular in Digital TV set-top box systems as part of the rapid merger of broadcast TV and web content. This is likely to be where JavaScript becomes a fairly dominant tool for developing interactive TV content.

The language continues to evolve and penetrate new markets and systems. The core language has been standardized and several bindings are also reasonably stable through the efforts of the W3C, but there is still much to be done to ensure the language operates consistently across browsers. Now that Netscape and MSIE are implementing the standards-based features in similar ways for the most part, the differences between the two browsers are being squeezed into more esoteric areas such as sidebars and visual transition filters. Events still have some way to go, and it will be a while before DOM-style control is completely supported. There are also several new DOM components in level 2 and extensions in level 3.

See the web reference for the article by Brendan Eich on "*The Birth of JavaScript*".

See also:	DOM, ECMA, ECMAScript, JavaScript language

Cross-references:

ECMA 262 edition 2 –section 4

ECMA 262 edition 3 –section 4

Web-references:

http://home.netscape.com/comprod/columns/techvision/innovators_be.html

P object (Object/HTML)

An object that encapsulates a paragraph delimited by a <P> tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0		
Inherits from:	`Element` object		
JavaScript syntax:	IE	*myP* = *myDocument*.*all*.*anElementID*	
	IE	*myP* = *myDocument*.*all*.*tags*("*P*")[*anIndex*]	
	IE	*myP* = *myDocument*.*all*[*aName*]	
	-	*myP* = *myDocument*.*getElementByID*(*anElementID*)	
	-	*myP* = *myDocument*.*getElementsByName*(*aName*)[*anIndex*]	
	-	*myP* = *myDocument*.*getElementsByTagName*("P")[*anIndex*]	
HTML syntax:	<P>,<P> ... </P>		
Argument list:	*anIndex*	A reference to an element in a collection	
	aName	An associative array reference	
	anElementID	The `ID` value of an `Element` object	
Object properties:	`align`		
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`		

The <P> tag is a block-level tag. That means that it forces a line break before and after itself.

The DOM level 1 specification refers to this as a `Paragraph` object.

See also:	`Element` object

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
align	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	1 +	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

.align (Property)

The alignment of the paragraph object with respect to its parent object.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myP*.align

The alignment of the `P` object with respect to its containing parent object is defined in this property. The expected and widely available set of alignment specifiers are available:

- `absbottom`
- `absmiddle`

- `baseline`
- `bottom`
- `center`
- `left`
- `middle`
- `right`
- `texttop`
- `top`

.pac (File extension)

Proxy lookup conversion file.

This is a script container for a small and compact JavaScript function that returns a computed value indicating whether to proxy-serve a URL or not.

See also:	Proxies, `proxy.pac`

Cross-references:

Wrox *Instant JavaScript* –page –58

package (Reserved word)

Reserved for future language enhancements.

See also:	`export`, `import`, Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Packages (Property)

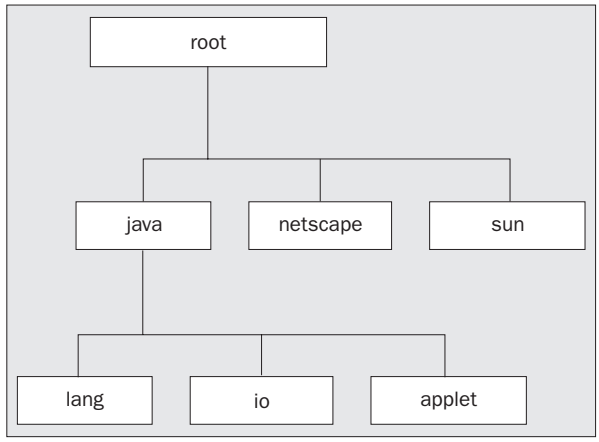
An alias for the `window.Packages` property.

Availability:	JavaScript –1.1 Netscape –3.0 Opera –3.0	
Property/method value type:	`JavaPackage` object	
JavaScript syntax:	N	*`myWindow`*`.Packages`
	N	`Packages`

This property contains a read-only reference to a JavaPackage that sits at the top of the Java package hierarchy, the root node of the tree. A JavaPackage is a container for other JavaPackage objects and JavaClass objects. By default, the Netscape Navigator browser will have three packages belonging to this top-level node (java, sun, and Netscape), and these will contain other packages. There may be additional externally, supplied packages over and above these three default items.

It is understandable that Microsoft does not go very far with support for Java other than being able to run a Java applet. There is no access to Java packages directly from script in the same way. You can run a Java Applet and interact with that, but its not quite the same thing.

Perhaps because it is so platform-specific, not very many people have explored the use of packages very deeply, even on Netscape Navigator. That seems a shame because within a captive intranet situation, you might be able to accomplish some useful things by creating fragments of very powerful Java code and then calling them from JavaScript. There doesn't seem to be the lengthy start up time that's required to initiate an Applet either.



See also:	`Window.Packages`

Property attributes:

`ReadOnly`.

Packages.java (Java package)

A package containing a collection of generic Java classes maintained as a package.

Availability:	JavaScript –1.1 Netscape –3.0 Opera –3.0

An example of a Package reference is the Java Date class stored in the `java.util` package.

To access this from JavaScript you would use this kind of construction:

```
myJavaClass = Packages.java.util.Date;
```

That mode of access would yield a reference to the Class and would produce a `JavaClass` object in the JavaScript environment. To create an instance and yield an object of that class, use the class as a constructor. Like this:

```
myJavaObject = new Packages.java.util.Date;
```

That would create a `JavaObject` object in the JavaScript environment.

If necessary, you may want to access a collection of Classes, which is called a Package. Here is how to create a JavaScript environment's `JavaPackage` object:

```
myJavaPackage = Packages.java.util;
```

See also:	`JavaClass` object, `JavaObject` object, `JavaPackage` object, `Window.java`

Packages.netscape (Java package)

A package containing a collection of Netscape-defined Java classes maintained as a package.

Availability:	JavaScript –1.1 Netscape –3.0 Opera –3.0

This is a code support for Java applets that use LiveConnect to access JavaScript from within the Java context.

See also:	`JavaClass` object, `JavaObject` object, `JavaPackage` object, `Window.netscape`

Packages.netscape.javascript (Java package)

A package containing support for Java code that needs to integrate with JavaScript via LiveConnect.

Availability:	JavaScript –1.1 Netscape –3.0

Refer to:

Java calling JavaScript

Packages.netscape.plugin (Java package)

A package containing support for applets and plugins that integrate with JavaScript via LiveConnect.

Availability:	JavaScript –1.1 Netscape –3.0
See also:	`Plugin.description` `Plugin` object

Packages.sun (Java package)

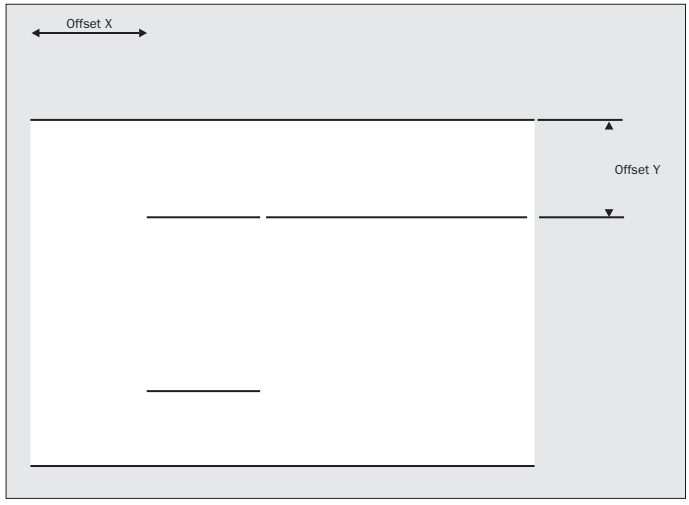
A package containing a collection of Sun Microsystem- defined Java classes maintained as a package. This includes some Sun Java security support as well.

Availability:	JavaScript –1.1 Netscape –3.0 Opera –3.0
See also:	`JavaClass` object, `JavaObject` object, `JavaPackage` object, `Window.sun` Java calling JavaScript

pageXOffset (Property)

An alias for the `window.pageXOffset` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myWindow*`.pageXOffset`
	N	`pageXOffset`



Property attributes:

`ReadOnly`.

Refer to:

`Window.pageXOffset`

pageYOffset (Property)

An alias for the window `pageYOffset` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myWindow*.pageYOffset
	N	pageYOffset

Property attributes:

`ReadOnly`.

Refer to:

`Window.pageYOffset`

ParamElement object (Object/HTML)

An object that encapsulates one of the parameters passed to an `OBJECT` object from its `<PARAM>` tags.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myParam* = *myDocument*.all.*anElementID*
	IE	*myParam* = *myDocument*.all.tags("PARAMETER")[*anIndex*]
	IE	*myParam* = *myDocument*.all.[*aName*]
	-	*myParam* = *myDocument*.getElementByID(*anElementID*)
	-	*myParam* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myParam* = *myDocument*.getElementsByTagName("PARAMETER")[*anIndex*]
HTML syntax:	`<PARAM>`	

Argument list:	*anElementID*	The ID value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Object properties:	name, type, value, valueType	
Event handlers:	onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp	

This is a new object introduced with the DOM specification. Its full name is HTMLParamElement. It can only exist as a child element within a block structured <OBJECT> tag.

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
name	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
type	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
value	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
valueType	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onHelp	-	5.0 +	-	5.0 +	-	-	-	Warning
onKeyDown	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning

Inheritance chain:

Element object, Node object

ParamElement.name (Property)

The name of the parameter passed to the OBJECT object.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myParamElement*.name

ParamElement.type (Property)

The type of parameter being passed to an `OBJECT` object when the `valueType` property is set to "`ref`".

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myParamElement.type`*

ParamElement.value (Property)

The data value being passed to the `OBJECT` object in the parameter.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myParamElement`*`.value`

ParamElement.valueType (Property)

The type of the value data that is being passed.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myParamElement`*`.valueType`

Parameter (Definition)

The formal description of a function interface.

The arguments that a function expects to be passed when it is called are described as a set of formal parameters when it is declared.

The parameters are called arguments from within the function when it is executed.

Since JavaScript is weakly data-typed, you do not need to specify the data type of the parameters in the function declaration. Nor are the arguments tested for compliance with any data type when the function is invoked. Rather the values of any arguments are coerced or cast as they are used according to the context in which they are referred to. This makes it very important that you take care with the type and value of parameters you are passing to functions. In particular with numeric and string values.

See also:	Argument, Argument list, `Arguments` object, Definition, Function, `Function` object, `function( ... ) ...`, `Function.arguments[]`, Reserved word, Scope chain

parent (Property)

An alias for the `window.parent` property.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Window` object	
JavaScript syntax:	-	*myWindow*`.parent`
	-	`parent`

Property attributes:

`ReadOnly`.

Refer to:

`Window.parent`

Parentheses ( ) (Delimiter)

A precedence of execution control mechanism.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0

Expression evaluation order is controlled by enclosing expressions in parentheses.

See also:	Grouping operator ( )

Cross-references:

ECMA 262 edition 2 –section –11.1.4

ECMA 262 edition 3 –section –11.1.6

parseFloat() (Function/global)

Parse a string to extract a floating-point value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`parseFloat(`*`aNumericString`*`)`
Argument list:	*`aNumericString`*	A meaningful numeric value

The `parseFloat()` function returns a numeric value, unless the string cannot be resolved to a meaningful value in which case `NaN` is returned instead.

It produces a number value dictated by interpreting the contents of the string as if it were a decimal literal value. During conversion `parseFloat()` ignores leading white space characters so you don't have to remove them from the string before conversion takes place.

Note that `parseFloat()` will only process the leading portion of the string. As soon as it encounters an invalid floating-point numeric character it will assume the scanning is complete. It will then silently ignore any remaining characters in the input argument.

See also:	`Cast` operator, `Global` object, `parseInt()`, String concatenate (+)

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.1.2.3

ECMA 262 edition 3 –section –15.1.2.3

parseInt() (Function/global)

Parse a string to extract an integer value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`parseInt(`*`aNumericString`*`,` *`aRadixValue`*`)`
Argument list:	*`aNumericString`*	A string that comprises a meaningful numeric value
	`aRadixValue`	A numeric value indicating the radix for conversion

The `parseInt()` function produces an integer value dictated by interpreting the string argument according to the specified radix. It can happily cope with hexadecimal values specified with the leading 0x or 0X notation. During conversion `parseInt()` will remove any leading whitespace characters. You don't need to do that to the string before parsing it.

Note also that `parseInt()` may only interpret the leading portion of a string. As soon as it encounters an invalid integer numeric character it will assume the scanning is complete. It will then silently ignore any remaining characters in the input argument.

Typical radix values are:

- 2 –Binary
- 8 –Octal
- 10 –Decimal
- 16 –Hexadecimal

The result of this function call is an integer value, unless the string cannot be resolved to a meaningful value in which case `NaN` is returned instead.

See also:	`Cast` operator, `Function` property, `Global` object, `parseFloat()`, String concatenate (+)

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 –section –15.1.2.2

ECMA 262 edition 3 –section –15.1.2.2

Password object (Object/DOM)

A text field in a form that echoes bullets instead of the typed character. Behaves as if it were a text cell but you cannot see what was typed.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0
Inherits from:	`Input` object
JavaScript syntax:	- *myPassword* = *myDocument*.*aFormName*.*anElementName* - *myPassword* = *myDocument*.*aFormName*.elements[*anItemIndex*] IE *myPassword* = *myDocument*.all.*anElementID* IE *myPassword* = *myDocument*.all.tags("INPUT")[*anIndex*] IE *myPassword* = *myDocument*.all[*aName*] - *myPassword* = *myDocument*.forms[*aFormIndex*].*anElementName* - *myPassword* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*] - *myPassword* = *myDocument*.getElementByID(*anElementID*) - *myPassword* = *myDocument*.getElementsByName(*aName*)[*anIndex*] - *myPassword* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
HTML syntax:	`<INPUT TYPE="password">`
Argument list:	*anIndex* — A valid reference to an item in the collection *aName* — The `name` attribute of an element *anElementID* — The `ID` attribute of an element *anItemIndex* — A valid reference to an item in the collection *aFormIndex* — A reference to a particular form in the forms collection
Object properties:	`maxLength, readOnly, size, type, value`
Object methods:	`handleEvent(), select()`
Event handlers:	`onAfterUpdate, onBeforeUpdate, onBlur, onChange, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onSelect, onSelectStart`

Many properties, methods, and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

Event-handling support via properties containing function objects was added to `Password` objects at version 1.1 of JavaScript.

Some implementations will not allow JavaScript to read the password string that the user has entered. This is good. You might imagine otherwise on the grounds that you'd expect then that JavaScript then won't be able to validate the password. If you think about this for a minute you'll realize that View Source in a web browser exposes your entire security checking regime. Its actually quite sensible to disallow JavaScript from inspecting the password field. But then, all the user has to do is run different browser –one that does access the contents of the password field.

Realistically the validation of the password can only be done back at the server anyway and other than some simple range checking in the client end access to password values from JavaScript is of doubtful use.

See also:	`Element` object, `Form.elements[]`, `FormElement` object, `Input` object, `Input.accessKey`, `onBlur`, `onChange`, `onFocus`, `Password.handleEvent()`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`maxLength`	-	3.0 +	-	4.0 +	-	-	-	-
`readOnly`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
`size`	-	3.0 +	-	4.0 +	-	-	-	Warning
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-
`select()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onChange`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelect	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page –645

Password.handleEvent() (Method)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	undefined	
JavaScript syntax:	N	*myPassword*.handleEvent(*anEvent*)
Argument list:	*anEvent*	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event-handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`handleEvent()`, `Password` object

Password.maxLength (Property)

The maximum length allowed for a password entry field.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myPassword*.maxLength

This defines the maximum number of characters that can be entered into the password field. The browsers differ in how they handle this value. Some will warn the user with a beep or flash on the screen, others simply stop accepting keystrokes when this number of characters has been entered.

See also:	`Input.maxLength`, `TextCell.maxLength`

Password.readOnly (Property)

Set to `true` if the Password field cannot be changed.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean value`	
JavaScript syntax:	IE	*`myPassword`*`.readOnly`

The password value is defined but cannot be changed by the user. The user should not be able to see the password on the screen; however, you might be able to view source to see the value in the HTML document source. Don't rely on this for hiding passwords in pages and assume that users will discretely ignore them. You might as well publish the password in the page heading.

See also:	`Input.readOnly`, `TEXTAREA.readOnly`, `TextCell.readOnly`

Property attributes:

`ReadOnly`.

Password.select() (Method)

All text in the password text entry cell is selected and can be cut and pasted by the user.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
JavaScript syntax:	-	*`myPassword`*`.select()`

If the browser supports a `Selection` object or `TextRange` objects, you may then be able to access the selected text using JavaScript. Of course in a form object, the text of the whole object can also be accessed but this may not be what was selected because the user may select all or part of a page and that selection may span several form elements or select only part of a form element.

See also:	`Input.select()`, `Selection` object

Password.size (Property)

The width of the password text box measured in characters.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myPassword*`.size`

This is an approximate measure at best. You cannot be sure how wide this box really needs to be when using a proportionally spaced font in it. The browser will size the box close to an optimal size to cope with the specified number of characters.

Warnings:

- It can be quite distracting if the box size is too small to accommodate the `maxLength` number of characters. This can leave the user having to do some cumbersome select actions with the mouse or use arrow keys to reveal the hidden parts of the textual content of the box.

See also:	`Input.size,TextCell.size`

Password.type (Property)

The type value for the `<INPUT>` object that describes the password text entry field in a form.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myPassword*`.type`

The type value for a password input text cell is always "`password`". This value is necessary to determine the type of form element because this object is really an instance of the `Input` class and not the `Password` class. There is actually no `Password` class.

See also:	`Input.type`

Property attributes:

`ReadOnly`.

Password.value (Property)

The user-entered value for the password text cell.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myPassword*.value

This is the string value that is sent back to the web server when the form is submitted. It is difficult to store script-driven values in this property or to define default values in the HTML document source without revealing the password value to the user. More experienced users will be able to work out any default password values if you hide them.

See also:	`Input.value`

Pattern matching (Definition)

Part of the regular expression support in some implementations of JavaScript.

Refer to:

`RegExp` pattern

PDF (Standard)

A de facto standard for portable documents, which is owned by Adobe Inc.

JavaScript is used inside Acrobat 4.0 as a forms-handling language. This provides a scripting environment in which you can manipulate the form data whose layout is defined by PostScript but whose content can then be 'activated' by JavaScript. There are a few minor limitations imposed due to the fact that Acrobat is not a web browser. There are also several additional objects provided to support the PDF forms environment.

See also:	Host environment, Platform, Script execution

Web-references:

http://www.pdfzone.com/pdfs/PDFSPEC13.PDF

Perl Connect (Product)

A mechanism for communicating between JavaScript and Perl scripts.

This is a means of allowing JavaScript code to call a Perl interpreter from inside the JavaScript environment. It is functionally similar to using the `eval()` method but instead of evaluating JavaScript, some Perl source is passed instead.

Web-references:

http://lxr.mozilla.org/mozilla/source/js/src/perlconnect/README.html

personalbar (Property)

An alias for the `window.personalbar` property.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Bar` object	
JavaScript syntax:	-	*myWindow*`.personalbar`
	-	`personalbar`

See also:	`Bar` object

Property attributes:

`ReadOnly`.

Refer to:

`Window.personalbar`

Pitfalls (Advice)

There are many pitfalls for the unwary in JavaScript.

Although JavaScript is a very forgiving language, over the years it has become large and complex. There are certainly ways in which the unwary can be caught out. Even long experienced script developers are taken unawares from time to time.

We present warning sub-sections summarized under each topic. Several pitfalls are so large as to warrant a topic of their own. These are some examples:

- Accidentally closing `</SCRIPT>` tags within the script itself
- Interpreting punctuation as tags
- Quotes can be a problem
- Hiding scripts from old browsers
- Line breaks in `document.write()` methods
- Browser detection for handling layers

The topic names (and hence their lexical order in this reference) are summarized in the See Also list:

See also:	`</SCRIPT>`, Deprecated functionality, Escaped JavaScript quotes in HTML, Hiding scripts from old browsers, HTML entity escape, JavaScript entity, Newlines are not ` ` tags, Off by one errors

Pixelate() (Filter/transition)

A transition effect with the appearance of a coarse pixelated dissolve.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

filter – `pixelate()`

Pixelate() (Filter/visual)

An effect that simulates the pixelation achieved when lowering the display resolution of an image.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `Pixelate()`

Pkcs11 object (Object/Navigator)

A hitherto undocumented object type supported by Netscape.

Availability:	JavaScript –1.2 Netscape –4.04	
JavaScript syntax:	N	*myPkcs11* = *myWindow*.pkcs11
	N	*myPkcs11* = pkcs11

The `Pkcs11` object is part of the security model built into Netscape Navigator. It is otherwise known as Cryptoki and is provided by RSA Data Security, Inc.

They implement a C language API that has now been mapped to Java as well. According to the Netscape web site, it is not a fully-fledged object-oriented API, but can be readily understood by programmers already familiar with Cryptoki.

According to the release notes, Netscape 4.04 added support for the FORTEZZA PKCS#11 module for making use of the FORTEZZA Crypto Card and FORTEZZA cryptographic algorithms (KEA and Skipjack) when using SSL and S/MIME. Although a link was provided for more details, it appears that the support documents may have been moved or deleted.

See also:	Cryptoki, `Window.pkcs11`

Property attributes:

`ReadOnly`.

Web-references:

http://developer.netscape.com/support/faqs/pkcs_11.html

PLAINTEXT object (Object/HTML)

An object that encapsulates a deprecated `<PLAINTEXT>` tag.

Availability:	JScript –3.0 Internet Explorer –4.0 Deprecated	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myPLAINTEXT* = *myDocument*.all.*anElementID*
	IE	*myPLAINTEXT* = *myDocument*.all.tags("PLAINTEXT")[*anIndex*]
	IE	*myPLAINTEXT* = *myDocument*.all[*aName*]
	-	*myPLAINTEXT* = *myDocument*.getElementByID(*anElementID*)
	-	*myPLAINTEXT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myPLAINTEXT* = *myDocument*.getElementsByTagName ("PLAINTEXT")[*anIndex*]
HTML syntax:	`<PLAINTEXT>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

Platform (Definition)

An environment is built to run on a platform that describes an OS and a hardware configuration.

JavaScript is becoming available on a variety of platforms:

- Web browsers
- TV set-top boxes
- Mobile phones
- Portable documents

Each of these uses JavaScript in a different way, although the core interpreted functionality is the same. Generally the differences will be in the area of the object model they support as part of the host environment. This is added to the core object model defined in the ECMA standard.

See also:	CGI Driven JavaScript, Desktop JavaScript, Embedded JavaScript, File extensions, Host environment, iCab, Internet Explorer, Netscape , OpenTV, Opera, PDF, Script execution, Shell Scripting with JavaScript, Web browser, WebTV, WScript

Cross-references:

Wrox *Instant JavaScript* –page –2

Wrox *Instant JavaScript* –page –5

Plugin compatibility issues (Definition)

Not all plugins behave the same.

The area where you may experience the most difficulty with plugins is to do with video and media. There are now three major competing technologies and many other minority plugins that can play audio and video.

If you just concentrate on the main players, you still have a lot of work to do. These are:

- Apple QuickTime
- Progressive Networks Real Media
- Windows Media Services

Functionally they are all very similar. Some of the protocols are shared between them and they all deliver multiple streams to a player plugin.

Other plugins that you'll encounter will be:

- Macromedia Shockwave
- Macromedia Flash
- LiveAudio
- Beatnik
- PDF viewer
- VRML viewer

Beyond that, some prefer an `<EMBED>` while others prefer `<OBJECT>` tags. This is platform- and browser-dependant so on one browser a plugin may `<EMBED>` and on another, the same plugin will work best in an `<OBJECT>` tag.

Be aware that using the same plugin in the same browser and implementing it as an `<OBJECT>` or an `<EMBED>` may mean that certain functionality is only available in one or the other.

Given that you have got the plugin working with the correct tag and the features you need are available, the call-back message they use are not the same, nor are the method calls you can make on the plugin. Indeed, for some plugins, you may only be able to talk to them from JavaScript with the very latest version. Older versions simply ignore the JavaScript messages.

Given that you now have your plugin running, in the right tags, and can talk to it, the functionality is different. Some plugins may allow the clip to be played backwards while others won't. Some may let you pause and play with different method calls, while at least one provides only a `play_pause` command. In that scenario, you can play or pause the clip but you won't actually know what state it is in.

Warnings:

- Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the `EMBED` class. In Netscape they are objects commonly referred to as belonging to the `Plugin` class.
- There is additional confusion in that there is a `plugins[]` array that belongs to the document and another than belongs to the navigator object. They both contain collections of objects but of different types. This is further confused by the fact that the `document.plugins[]` array is another name for the `document.embeds[]` array.
- Due to this confusing situation, the best recommendation is that we refer to `document.embeds[]` and `navigator.plugins[]` and quietly ignore the `document.plugins[]` array. Furthermore we shall refer to `Plugin` objects as being something the browser can use to play embedded content and, `Embed` objects, will be an instance of a plugin that is alive and running in a document.

See also:	Compatibility, Plugin events

Plugin events (Definition)

The events that are triggered by plugins are commonly referred to as callbacks.

There are basically three kinds of plugins available with web browsers. These are:

- Microsoft `<OBJECT>` tags enclosing ActiveX objects
- Netscape `<EMBED>` tags enclosing browser plugins
- Java `<APPLET>` tags enclosing Java applets

In Netscape Navigator, the LiveConnect mechanisms can run specific functions by name. This callback mechanism is available to `<EMBED>` and `<APPLET>` plugins, but it isn't clear to what extent this is supported by `<OBJECT>` plugins so you may need to experiment with your browser to see what works and what doesn't. Use the `<EMBED>` and `<APPLET>` functionality as a guideline because the plugin handlers are all likely to share some code. Even so, it requires the Java VM (Virtual Machine) to be started for it to work. LiveConnect also requires that the `<APPLET>` tag contains the `MAYSCRIPT` attribute to give the applet permission to communicate with JavaScript.

In MSIE, the callbacks are managed via ActiveX mechanisms. Again, an ActiveX object can call a named JavaScript function.

Warnings:

- The distinctions between these different plugin architectures become increasingly blurred and basically all of their functionality could probably have been provided with a single mechanism had the browser manufacturers worked together more co-operatively and been less concerned with carving out territory and gaining market share at each other's expense.
- Interacting with plugins using JavaScript is possibly one of the least portable and most frustrating things to develop solutions for. The competing plugin suppliers have also utterly failed to develop common API calls for media players such that you need to treat each one as a special case even if you only want to play, pause, and stop video clips under JavaScript control.

- Neither of the mainstream browsers supports the other's chosen technology fully or reliably. It is not uncommon to find MSIE crashes on Windows with <EMBED> style plugins. Netscape Navigator does not support <OBJECT> plugins at all well, and ActiveX code crashes a Macintosh horribly when either browser tries to run it. That's understandable since it's usually X86 machine code and right now Power PC processors don't like it. The latest MSIE version 5.0 for the Macintosh provides a switch to disable ActiveX plugins altogether. Even Microsoft Windows Media Player on Macintosh is supported by way of an <EMBED> tag. Real player is recommended to be used as an <OBJECT> plugin on MSIE for Windows although it mostly works the same as an <EMBED>. There are some JavaScript API calls that are not available in both modes though.
- This means for example to embed video into a page, even with the same kind of video, you basically have to implement four different containers to support MSIE and Netscape on Macintosh and Windows. Given that, there are now other browsers that can cope with plugins. That increases the number of varieties of HTML page content that needs to be created. Now that QuickTime, Windows Media Services and Real Media are all contending for market share, they all require different plugins and that multiplies your problem threefold. Now, it is likely that some of the platform/media combinations will be similar enough to share the same code. Even then, it still leaves you with something like six radically different ways that you will need to construct the plugin container if you want to do anything sophisticated and JavaScript driven.

See also:	<EMBED>, Event, LiveConnect, Plugin compatibility issues, Plugin object

Cross-references:

Wrox *Instant JavaScript* –page –55

Plugin object (Object/browser)

An object representing a plugin.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
JavaScript syntax:	-	*myPlugin* = document.plugins[*anIndex*]
	N	*myPlugin* = navigator.plugins[*anIndex*]
	-	*myPlugin* = *myPluginArray*[*anIndex*]
HTML syntax:	<APPLET><EMBED><OBJECT>	
Argument list:	*anIndex*	A reference to an element in a collection
Object properties:	description, filename, length, name	
Object methods:	isActive(), refresh()	

Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the EMBED class. In Netscape Navigator they are objects commonly referred to as belonging to the Plugin class although they are really implemented as JavaObject objects. In MSIE, this is an ActiveX object.

There is additional confusion in that there is a `plugins[]` array that belongs to the document and another than belongs to the navigator object. They both contain collections of objects but of different types. This is further confused by the fact that the `document.plugins[]` array is another name for the `document.embeds[]` array.

Because of this confusing situation the best recommendation is that we refer to `document.embeds[]` and `navigator.plugins[]` and quietly ignore the `document.plugins[]` array. Furthermore we shall refer to `Plugin` objects as being something the browser can use to play embedded content and an `Embed` object will be an instance of a plugin that is alive and running in a document.

Warnings:

- Do not confuse `Plugin` and `Embed` objects with one another. Plugin objects are owned by the `navigator.plugins` array. `Embed` objects are owned by the `document.embeds` array.

See also:	`<EMBED>`, `Collection` object, `Embed` object, `EmbedArray` object, Glue code, Java, `Navigator.plugins[]`, Plugin events, `PluginArray` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`description`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	ReadOnly
`filename`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	ReadOnly
`length`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	Warning, ReadOnly
`name`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
`isActive()`	1.3 +	-	4.7 +	-	-	-	-
`refresh()`	1.1 +	-	3.0 +	-	3.0 +	-	-

Plugin.description (Property)

The descriptive text that a plugin yields when requested to do so.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myPlugin`*`.description`

The content of this string depends on what the plugin developer coded. It might be helpful for debugging and maybe you could present a list of installed plugins to the user. The standardization of the plugin descriptions is likely to be worse than that of the `Navigator.userAgent` string and therefore trying to formulate rules for parsing these descriptions is going to be difficult.

Property attributes:

`ReadOnly`.

Plugin.filename (Property)

The filename that a plugin is stored in.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myPlugin`*`.filename`

This property may help in diagnosing plugins that are not working. If you can find out where in the file system it has installed, you may be able to go and replace it with an up-to-date copy if it is failing to work.

Property attributes:

`ReadOnly`.

Plugin.isActive() (Method)

A means of detecting whether an applet or plugin is still active.

Availability:	JavaScript –1.3 Netscape –4.7	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*`myPlugin`*`.isActive()`

Refer to:

LiveConnect

Plugin.length (Property)

The number of MIME types supported by the plugin.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myPlugin`*`.length`

Warnings:

- Be careful not to confuse this property with the `document.plugins.length` and `navigator.plugins.length` properties.
- The `Plugin.length` property is the number of MIME types that a plugin can respond to.
- The `document.plugins.length` property gives a count of the number of plugins there are embedded into a document.
- The `navigator.plugins.length` property is the number of plugin support modules that are installed and available to the browser.

See also:	`Collection.length`, `PluginArray.length`

Property attributes:

`ReadOnly`.

Plugin.name (Property)

This corresponds to the `NAME` attribute of the tag that contains the plugin.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myPlugin*.name
HTML syntax:	`<EMBED NAME="`*aName*`">`	
Argument list:	*aName*	A name to identify the plugin object

Objects are identified either by the `NAME="..."` HTML tag attribute or by the `ID="..."` HTML tag attribute.

Netscape shows a marginal preference for the `name` property while MSIE seems slightly better disposed towards the `ID` property. However in many cases, both browsers support either technique and in some cases will locate items named with either tag as if they existed in a single namespace.

See also:	`NAME="..."`

Property attributes:

`ReadOnly`.

Plugin.refresh() (Method)

A method to reload the plugin.

Availability:	JavaScript –1.1 Netscape –3.0 Opera –3.0	
JavaScript syntax:	N	*myPlugin*.refresh()

You need to refresh a plugin if you have installed it since the browser was started. This method is sometimes placed in pages that use plugins so that a refresh is forced every time the page is loaded. This is not strictly necessary but has been found to prevent some strange run-time errors in older versions of Netscape Navigator.

PluginArray object (Object/browser)

A collection of plugin modules that the browser can use to playback embedded content.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
JavaScript syntax:	IE	*myPluginArray* = document.plugins
	N	*myPluginArray* = navigator.plugins
Object properties:	length	
Object methods:	item(), refresh()	

Netscape and MSIE encapsulate plugin/embedded objects in a different way. In MSIE they are objects of the EMBED class. In Netscape Navigator they are objects commonly referred to as belonging to the Plugin class although they are really implemented as JavaObject objects. In MSIE, this is an ActiveX object.

There is additional confusion in that there is a plugins[] array that belongs to the document and another than belongs to the navigator object. They both contain collections of objects but of different types. This is further confused by the fact that the document.plugins[] array is another name for the document.embeds[] array.

Due to this confusing situation, the best recommendation is that we refer to document.embeds[] and navigator.plugins[] and quietly ignore the document.plugins[] array. Furthermore we shall refer to Plugin objects as being something the browser can use to play embedded content and an Embed object will be an instance of a plugin that is alive and running in a document.

Warnings:

- Beware of confusion between `document.plugins` and `navigator.plugins`. One relates to the plugins currently used in the document while the other lists the plugins currently available and supported by the browser.
- In Netscape 4.7 for Macintosh, there is a strange enumeration problem. Immediately after starting the Netscape browser, when you enumerate the properties of the `netscape.plugins PluginArray` object, it appears to have no properties at all. If you explicitly ask for the `length` property, you will get a value. During investigation, it returned the value 8 but this will depend on the number of plugins you have installed.
- Now, this suggests that you should be able to access the plugins individually by index number. As soon as you access one of the plugins by its numeric index, Netscape Navigator also adds an entry using the plugin name so you can access it associatively. However, you can also enumerate the item you just created until the browser clears the array (probably when the application exits). So, although you cannot enumerate the plugins from cold, you can enumerate the ones that you have accessed by index value.
- Sending a refresh message to a plugin object also allows it to be enumerable. Based on this idea, a short fragment of code is given in the example that will force all the plugins to be added to the collection as associative items, which can then be enumerated.
- MSIE allows the plugins to be enumerated and the length property is also enumerable. However, the plugins can only be accessed by their numeric index.
- To make this properly portable, execute the bug fix code and access plugins by their numeric index and your scripts should then be reasonably portable.

Example code:

```
// Execute this in Netscape Navigator to fix the
// navigator.plugins enumeration bug
for(ii=0; ii<navigator.plugins.length; ii++)
{
   navigator.plugins[ii].refresh;
}
```

See also:	`Collection` object, `EmbedArray` object, `Navigator.plugins[]`, `Plugin` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`length`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	Warning, ReadOnly

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
`item()`	-	3.0 +	-	4.0 +	-	-	-
`refresh()`	1.1 +	-	3.0 +	-	3.0 +	-	-

PluginArray.item() (Method)

An item selector for accessing a single plugin within the collection.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Plugin` object	
JavaScript syntax:	IE	*`myPluginArray`*`.item(`*`anIndex`*`)`
	IE	*`myPluginArray`*`.item(`*`aSelector`*`)`
	IE	*`myPluginArray`*`.item(`*`aSelector`*`, `*`anIndex`*`)`
Argument list:	*`anIndex`*	A zero based index into the collection
	`aSelector`	A textual value that selects all matching objects

Refer to:

`Collection.Item()`

PluginArray.length (Property)

The number of plugin objects currently supported by the browser.

Availability:	JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`document.plugins.length`
	-	`navigator.plugins.length`

Warnings:

- Be careful not to confuse this property with the `Plugin.length` property.
- The `Plugin.length` property is the number of MIME types that a plugin can respond to.
- The `document.plugins.length` property gives a count of the number of plugins there are embedded into a document.
- The `navigator.plugins.length` property is the number of plugin support modules that are installed and available to the browser.

See also:	`Collection.length`, `Plugin.length`

Property attributes:

`ReadOnly`.

PluginArray.refresh() (Method)

Refresh all the plugins in the current page.

Availability:	JavaScript –1.1 Netscape –3.0 Opera –3.0	
JavaScript syntax:	N	`document.plugins.refresh()`
	N	`navigator.plugins.refresh()`

See also:	`Navigator.plugins.refresh(),Plugin.refresh()`

Pointers (Overview)

JavaScript does not have pointers. On the other hand it does have good garbage collection.

Refer to:

Garbage collection

Polymorphic (Definition)

Operations that are workable on a variety of data types.

Refer to:

Type conversion

Portability (Definition)

The ability of your script to run on multiple platforms.

In general, JavaScript is highly portable. However, it is dubious as to whether a server-side script would be useful on the client side, so some measure of discretion is required when talking about portability. Even so, it is likely that many useful functions would work in different contexts although the way they are used may be different.

Portability of code needs to be considered at the outset of any development project. A script is portable to the extent that it can be used on a variety of platforms.

Generally, code in a function is more portable than an entire script-driven application. You might reuse functional blocks of code from one project to another. JavaScript facilitates this reuse due to its semi-object-oriented nature.

If you adhere strictly to the ECMAScript standard, then your script may port between compliant implementations with the minimum of difficulty.

Note that portability does not imply that a script you design to work client side is guaranteed to work meaningfully on the server side. There is the small matter of fitness for purpose and suitability.

However, it is not unreasonable to expect a script be able to run identically in all versions and flavors of web browsers. Realistically though, this is very difficult to accomplish because of the browser developers having added proprietary extensions to their implementations. They also increase the difficulty due to the amount of incomplete and incorrect support for standardized behavior. Furthermore the standard is ambiguous in a few areas leading to functional interpretations, which can vary from browser to browser.

For example, both MSIE and Netscape can scroll the contents of a window or frame. MSIE will scroll under script control whether the window has a scroll bar or not. Netscape Navigator will only scroll if a scroll bar is flagged `true` and is actually present.

Furthermore when scrolling is possible (on Netscape it can be accomplished without scroll bars by using layers), each browser scrolls in the opposite direction, as the scroll value is incremented.

Do not assume that expressions will evaluate the operands in the order of presentation. There are various ways to tokenize and interpret expressions and they can evaluate individual items in different sequence order, or in some cases may omit to evaluate an operand, when an early prediction of the outcome is possible.

Be wary of using non uniform character values that appear to display a certain character, which is not the correct one, defined in the Unicode character set. Rather, escape the character either with a Unicode escape or if it is appropriate for the context as an HTML character entity.

Be careful with quotes and escaping of them. If you escape a single quote using the HTML character entity, then some web browsers will un-escape the source text before interpretation. This breaks:

```
myString = 'some text with a &#039; single quote';
```

the HTML character entity is decoded too soon, by the browser as the page is read into its memory cache. Then the script is interpreted, and the interpreter sees this:

```
myString = 'some text with a ' single quote';
```

The additional quote terminates the string too soon and the remainder is interpreted as code. This can have very unpredictable consequences. The apostrophe should be escaped like this:

```
myString = 'some text with a \' single quote';
```

Inevitably, there will be occasions where you need to factor your script to cope with machine-dependant behavior. You should collect the machine-dependant code into clearly identified functions. You can use the `eval()` function to great effect here by detecting the name of the browser and algorithmically generating the name of a function to call, which you can then invoke via an `eval()`.

See also:	Compatibility, HTML Character entity, Language codes, Undefined behavior

Cross-references:

Wrox *Instant JavaScript* –page –11

Positive value (+) (Operator/unary)

Indicate positive value or numeric cast a non-numeric value.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`+anOperand`
Argument list:	`anOperand`	A value that can reasonably be converted to a number

The operand is evaluated and converted to a numeric value.

A positive value is unchanged.

A negative value is unchanged.

A string value will be converted to a Numeric value and replaced in context.

Although this is classified as a unary operator, its functionality is really that of an additive operator.

The result will be the value of the operand, cast to a Numeric type.

See also:	Additive operator, Unary operator

Cross-references:

ECMA 262 edition 2 –section –11.4.6

ECMA 262 edition 3 –section –11.4.6

Postfix decrement (–) (Operator/postfix)

Decrement after access.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*anOperand*--
Argument list:	*anOperand*	A numeric value that can be decremented

The operand is decremented by one.

The operand is evaluated first and is then decremented when the evaluation is completed

The associativity is from right to left.

Refer to the Operator Precedence topic for details of execution order.

See also:	Associativity, Decrement value (--), Operator Precedence, Postfix expression, Postfix operator, Prefix decrement (--), Prefix expression

Cross-references:

ECMA 262 edition 2 –section –11.3.2

ECMA 262 edition 3 –section –11.3.2

Postfix expression (Operator/postfix)

Increment or decrement an operand after access.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

Postfix expressions operate on Left-Hand-Side (sometimes called LValue) expressions.

There are two postfix operators:

- ++ performs a numeric increment on the operand
- -- performs a numeric decrement on the operand

These can also be classified as additive operators and because they modify a value in place, they also imply that an assignment takes place as well.

See also:	Additive operator, Assignment operator, Decrement value (--), Expression, Increment value (++), Postfix decrement (--), Postfix increment (++), Prefix expression

Cross-references:

ECMA 262 edition 2 –section –11.3

ECMA 262 edition 3 –section –11.3

Postfix increment (++) (Operator/postfix)

Increment after access.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*anOperand*++
Argument list:	*anOperand*	An incrementable numeric value

The operand is incremented by one.

The operand is evaluated first and is then incremented when the evaluation is completed.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

See also:	Associativity, Decrement value (--), Increment value (++), Operator Precedence, Postfix expression, Postfix operator, Prefix expression

Cross-references:

ECMA 262 edition 2 –section –11.3.1

ECMA 262 edition 3 –section –11.3.1

Postfix operator (Definition)

Operators that are placed after the operand.

Postfix operators are placed immediately after their operands. The following postfix operators are supported:

Operator	Description
`( )`	Function arguments delimiter
`++`	Increment the operand
`--`	Decrement the operand
`.`	Object property delimiter
`[ ]`	Array element delimiter

The meanings of some of these operators may vary in other contexts.

See also:	Arithmetic operator, Array index delimiter ([ ]), Associativity, Object property delimiter (.), Operator, Operator Precedence, Postfix decrement (--), Postfix increment (++), Prefix operator, Unary operator

Cross-references:

Wrox *Instant JavaScript* –page –19

Power function (Definition)

A function that deals with powers of numbers.

There are several mathematical functions that deal with powers of numbers and some pre-computed constants that are useful in the same context. The following table summarizes JavaScript capabilities in this area:

Item	Type
`Math.pow()`	Function
`Math.sqrt()`	Function
`Math.SQRT1_2`	Constant
`Math.SQRT2`	Constant

See also:	Exponent-log function, Integer-value-remainder, `Math.pow()`, Mathematics, Trigonometric function

PRE object (Object/HTML)

An object that encapsulates the content of a <PRE> tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	Element object	
JavaScript syntax:	IE	myPRE = *myDocument*.all.*anElementID*
	IE	myPRE = *myDocument*.all.tags("PRE")[*anIndex*]
	IE	myPRE = *myDocument*.all[*aName*]
	-	myPRE = *myDocument*.getElementByID(*anElementID*)
	-	myPRE = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	myPRE = *myDocument*.getElementsByTagName ("PRE")[*anIndex*]
HTML syntax:	<PRE> ... </PRE>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	width	
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

The <PRE> tag is a block-level tag. That means that it forces a line break before and after itself.

See also:	Element object, KBD object, LISTING object, style.overflow

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

PRE.width (Property)

The width of a block of pre-formatted text.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myPRE*.width

This width value has a an upper range limit of 100, which represents 100% of the available width of the screen. This is not the usual percentage width of its parent containing element.

Pre-processing (Definition)

An extra processing step performed on the script source text before interpretation commences.

Programmers who come from a C programming background will be familiar with the C language pre-processor.

This is a stage in the compilation process that performs some processing on the source code prior to it being compiled.

Typically, this allows for the substitution of constant values, the expansion of macros (small fragments of code that are used very often) and the conditional inclusion or exclusion of code usually based on the platform being used.

Standard JavaScript does not support this functionality but several embedded interpreter products do, as does the ScriptEase interpreter. Now, it is available in JScript although you should limit its use to those situations where you know the script will be executed on a compliant system.

To avoid problems with non-compliant systems, you should place the pre-processor directives inside comment blocks. This at least will hide them from the platforms that don't support this capability. However, you need to be aware of how your script will behave if the pre-processor directives are ignored. This can become quite complex and may outweigh any advantages of using the pre-processor. You must make provision for this if your scripts need to run on Netscape Navigator for example, because it will cause an interpretation error when it sees the `@` symbol at the beginning of each directive.

The leading comment should be placed immediately in front of the `@` symbol. The trailing comment delimiter should have an additional `@` sign to indicate it matches the prefixing comment symbol. So the activation directive:

```
@cc_on
```

Should be coded like this:

```
/*@cc_on@*/
```

In a predictable environment, there are no concerns about portability and you need not use the comment encapsulation technique to hide the directives.

Placing the pre-processor directives inside quotes makes them ineffective.

Undefined values return the `NaN` value. There are no error reports when a directive that does not exist is used. However, placing a trailing `@` symbol without it being part of the comment hiding mechanism will generate a parsing error.

Warnings:

- Note that each implementation of a pre-processor is implementation-specific. There are as yet no standards for this and there are several alternative and mutually non-compliant alternatives. Some copy the C language pre-processor directives and commence with a hash symbol (#) while others use a completely different syntax altogether (for example, JScript with its `@` symbol).
- Although this capability was introduced in JScript 3.0, which was available in version 4 of MSIE, attempting to use the pre-processor directives may well crash a Macintosh version 4.5 MSIE browser. Some directives may work in certain restricted situations.
- MSIE version 5 for Macintosh exhibits some shortcomings in its implementation of the platform detection mechanisms.

See also: `@*/`, Pre-processing - `/*@ ... @*/`, Pre-processing - `@<variable_name>`, Pre-processing - `@_alpha`, Pre-processing - `@_jscript`, Pre-processing - `@_jscript_build`, Pre-processing - `@_jscript_version`, Pre-processing - `@_mac`, Pre-processing - `@_mc680x0`, Pre-processing - `@_PowerPC`, Pre-processing - `@_win16`, Pre-processing - `@_win32`, Pre-processing - `@_x86`, Pre-processing - `@cc_on`, Pre-processing - `@elif( ... ) ...`, Pre-processing - `@else ...`, Pre-processing - `@end`, Pre-processing - `@if( ... ) ...`, Pre-processing - `@set`

Pre-processing –/*@ ... @*/ (Delimiter)

A special form of the comment delimiters for enclosing pre-processor directives.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`/*@someDirectives@*/`
Argument list:	`someDirectives`	One or more pre-processor directives

This form of the comment delimiters is important when you need to use the JScript pre-processor directives. Enclosing them in a comment block hides them from non-compliant browsers and script interpreters.

The pre-processor directive has the following general format:

```
@<some_keyword>
```

To hide it within comments, you need to modify it so it resembles this general form:

```
/*@<some_keyword> @*/
```

There are special requirements for enclosing entire blocks of code when the conditional inclusion directives are used. Refer to the `@if` topic for more details.

It seems to be convention to place a space character in front of the closing `@*/` comment delimiter. This may not be strictly necessary for functional reasons but aids the readability of the directives when placed into portable code.

See also:	`@*/`, Pre-processing, Pre-processing - `@<variable_name>`, Pre-processing - `@_alpha`, Pre-processing - `@_jscript`, Pre-processing - `@_jscript_build`, Pre-processing - `@_jscript_version`, Pre-processing - `@_mac`, Pre-processing - `@_mc680x0`, Pre-processing - `@_PowerPC`, Pre-processing - `@_win16`, Pre-processing - `@_win32`, Pre-processing - `@_x86`, Pre-processing - `@cc_on`, Pre-processing - `@elif( ... ) ...`, Pre-processing - `@else ...`, Pre-processing - `@end`, Pre-processing - `@if( ... ) ...`, Pre-processing - `@set`

Pre-processing –@<variable_name> (Pre-processor)

A special pre-processor variable container.

Availability:	JScript –3.0 Internet Explorer –4.0
Property/method value type:	`User defined`

JavaScript syntax:	IE	`@aVariable`
	IE	`@aVariable=aValue`
Argument list:	`aValue`	A value to be assigned
	`aVariable`	A variable created with the `@set` directive

The pre-processor sub-system supports the definition and use of a special kind of variable. These are defined and modified with the `@set` pre-processor directive.

Once created, the variable can be used anywhere in the code, like this:

```
myString = "***" + @myvariable;
```

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@myvariable @*/
```

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_alpha (Pre-processor)

A pre-processor constant indicating whether the script is running in a DEC alpha workstation.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_alpha`

This pre-processor constant yields `true` when used on a DEC alpha processor and `NaN` otherwise.

Since MSIE only runs on Macintosh and Windows platforms, and a DEC alpha is not a Macintosh, there is an implication here that this will be `true` only when MSIE is running on Windows NT on a DEC alpha. It should also be the case that the `@_win32` directive returns `true` as well.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_alpha @*/
```

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_jscript (Pre-processor)

A pre-processor constant indicating whether the script is executing in a JScript interpreter.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_jscript`

This preprocessor constant yields `true` if the script is running in a genuine Microsoft JScript interpreter, and `NaN` if it is JavaScript but not JScript.

These directives may have been defined with a `@` symbol rather than a `#` symbol for reasons of consistency across a variety of Microsoft platforms. Therefore, this directive may be useful to be able to conditionally include script source that depends on the kind of interpreter being used.

For now this directive will always return the `true` value when used in the MSIE browser.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_jscript @*/
```

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_jscript_build (Pre-processor)

A pre-processor constant indicating the build version of the JScript environment.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`@_jscript_build`

This preprocessor constant returns the build number of the interpreter (but not the browser) in which the script is running.

For example, in version 5.0 of MSIE for Macintosh, the browser build number is 2022 but the JScript build number reported by this pre-processor directive is 3715.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_jscript_build @*/
```

Warnings:

- Oddly enough, in version 4.5 of MSIE, the value reported is also 3715 even though the build number of the browser is 0408 and is therefore much older.
- Because both versions of the browser were tested in the same machine, it is possible that by installing the MSIE 5 browser, some components of the browser are stored in the System folder. These may well have overwritten components that the version 4.5 browser was using and so you need to be aware of the possibility of MSIE browsers exhibiting odd behavior due to the way the application is factored into components. In fact this is confirmed by the fact that the `@_jscript_version` directive reports JScript 5 from within MSIE 4.5 which is not correct. Other aspects of the interpreter that interact with the browser core may exhibit JScript 3 functionality. So installing MSIE 5 on a Macintosh over the top of an MSIE 4.5 yields an interesting hybrid variant of the version 4.5 browser. Performing upgrades of your browser and JScript components on a Windows platform may yield similar hybrid variants.
- You should be careful that if you code for a version of something you should be testing that same thing. Don't test browser versions to conditionally execute JScript version-specific code.
- The Netscape Navigator browser code is contained more integrally within its own application space and you may be able to have several versions of that browser without any subtle interaction between low-level shared library modules.
- You may assume the version of the interpreter tells you something useful but don't assume any other implications regarding browser versions based on the interpreter version.

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_jscript_version (Pre-processor)

A pre-processor constant indicating the version number of the JScript interpreter.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`@_jscript_version`

This preprocessor constant provides the version number of the interpreter (but not the browser) in which the script is running.

For example, in version 5.0 of MSIE for Macintosh, the version of JScript expected is version 5.0 and you do get the value 5 reported by this directive.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_jscript_version @*/
```

Warnings:

- As is the case with the `@_jscript_build` directive, this one may be affected by the installation history of your workstation.
- Installing MSIE 5 on a Macintosh over the top of an MSIE 4.5 yields an interesting hybrid variant of the version 4.5 browser. Due to the component nature of Microsoft browsers and interpreters, the same is true on the Windows platform and you can very easily find the versions of browser and JScript integerpreter have diverged as a result of installing another application that may upgrade some shared components.
- You may assume the version of the interpreter tells you something useful but don't assume any other implications regarding browser versions based on the interpreter version or vice versa. Test the thing you need to know about and do not assume that the browser and interpreter are directly related to one another.
- The version history tables suggest they are related but this simply lists the versions of JScript that were shipped as part of the browser install kit for a fresh and complete installation.

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_mac (Pre-processor)

A pre-processor constant indicating whether the script is running in a Macintosh workstation.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_mac`

This directive should yield the value `true` when tested on any Macintosh system.

In MSIE version 5, it yields the value `NaN` which is what you would expect on a non-Macintosh system.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_mac @*/
```

Warnings:

- This does not appear to work in MSIE 5 for Macintosh.

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_mc680x0 (Pre-processor)

A pre-processor constant indicating whether the system contains a Motorola 68000 CPU.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_mc680x0`

This directive should yield the value `true` when tested on any older pre Power PC equipped Macintosh system and `NaN` when tested on a modern Power PC machine.

In MSIE version 5, it yields the value `NaN` regardless of the CPU, which is what you would expect on a non-68K equipped system.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_mc680x0 @*/
```

Warnings:

- This does not appear to work in MSIE 5 for Macintosh.

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_PowerPC (Pre-processor)

A pre-processor constant indicating whether the system contains a Motorola PowerPC CPU.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_PowerPC`

This directive should yield the value `true` when tested on any newer Power PC equipped Macintosh system and `NaN` when tested on an older 68K machine.

In MSIE version 5, it yields the value `NaN` regardless of the CPU, which is what you would expect on a non Power PC system.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_PowerPC @*/
```

Warnings:

- This does not appear to work in MSIE 5 for Macintosh.
- Be careful with capitalization on this directive; none of the others seem to require capital letters but this one does.

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_win16 (Pre-processor)

A pre-processor constant indicating whether the script is running in a 16 bit Windows environment.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_win16`

This directive should yield the value `true` when tested on any older 16 bit Windows system.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_win16 @*/
```

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_win32 (Pre-processor)

A pre-processor constant indicating whether the script is running in a 32 bit Windows environment.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_win32`

This directive should yield the value `true` when tested on any modern 32 bit Windows system.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_win32 @*/
```

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@_x86 (Pre-processor)

A pre-processor constant indicating whether the system contains an Intel X-86 series CPU.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`@_x86`

This directive should yield the value `true` when tested on any system equipped with an Intel X86 CPU. It should yield `NaN` on any non Intel system and always read `NaN` on a Macintosh.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@_x86 @*/
```

See also:	Pre-processing, Pre-processing - `/*@ ... @*/`

Pre-processing –@cc_on (Pre-processor)

A switch to activate the pre-processor phase of the script interpreter.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`@cc_on`

The pre-processor directives will not work unless they are first activated by placing this directive near the top of the script.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@cc_on @*/
```

See also:	Conditional code block, Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@elif(...) ... (Pre-processor)

An optional else-if pre-processor token.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`... @elif(`*`aCondition`*`) ...`
Argument list:	*`aCondition`*	A pre-processor supported condition test

This can be used in a conditional code block to indicate a supplementary conditional block.

Refer to the `@if()` topic for more details.

If you intend to hide this directive inside some comments, you must place the comment around the entire construct and not the individual pre-processor directives.

See also:	Conditional code block, Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@else ... (Pre-processor)

Part of the conditional code use directive.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`... @else ...`

This can be used in a conditional code block to indicate an alternative conditional block.

Refer to the `@if()` topic for more details.

If you intend to hide this directive inside some comments, you must place the comment around the entire construct and not the individual pre-processor directives.

See also:	Conditional code block, Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@end (Pre-processor)

Terminator for a conditional code block.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`... @end`

This should be used to terminate a conditional code block.

Refer to the `@if()` topic for more details.

If you intend to hide this directive inside some comments, you must place the comment around the entire construct and not the individual pre-processor directives.

See also:	Conditional code block, Pre-processing, Pre-processing – `/*@ ... @*/`

Pre-processing –@if(...) ... (Pre-processor)

Conditionally include a block of code.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`@if(`*`aCondition`*`)` *`someCode`* `@elif(`*`aCondition`*`)` *`someCode`* `@else` *`someCode`* `@end`
	IE	`@if(`*`aCondition`*`)` *`someCode`* `@elif(`*`aCondition`*`)` *`someCode`* `@end`
	IE	`@if(`*`aCondition`*`)` *`someCode`* `@else` *`someCode`* `@end`
	IE	`@if(`*`aCondition`*`)` *`someCode`* `@end`
Argument list:	*`aCondition`*	A condition that yeilds a Boolean value
	`someCode`	A block of code that is conditionally included

The `@if()` directive is very flexible, having two optional associated directives (`@elif()` and `@else`), which provide a variety of different configurations.

The simplest form is where a section of code is included or not. That would be organized like this:

```
@if(anExpression)

...

someCode

...

@end
```

If the expression yields a `true` value then the code will be included, otherwise it will be skipped, as if it had been completely commented out.

The next simplest form is to place an alternative section of code in the conditional section and have that used when the expression yields a `false` value. That would be laid out like this:

```
@if(anExpression)

...

someCode

...

@else

...

someOtherCode

...

@end
```

The result of the expression selects one block of code or the other.

A third and somewhat more complex configuration allows a series of conditions to be tested. This is somewhat like a switch tree although it is a little less elegant in its presentation. You can test for several conditions and include an appropriate block of code for the one that holds true. However, only one will be selected. This is accomplished with the `@elif()` directive. This is only tested when a prior `@if()` or `@elif()` test proves `false`.

Here is an configuration that tests for three possible conditions:

```
@if(anExpression)
...
someCode
...
@elif(anExpression)
...
someCode
...
@elif(anExpression)
...
someCode
...
@end
```

Note that in this form, there is no alternative block of code associated with an `@else` directive. One of these expressions must prove `true` for any code to be included. If none of them prove `true`, then the entire conditional block is ignored.

The final configuration provides a fall-back alternative code block and is constructed like this:

```
@if(anExpression)
...
someCode
...
@elif(anExpression)
...
someCode
...
@elif(anExpression)
...
someCode
```

```
...
@else
...
someOtherCode
...
@end
```

You may be able to nest these directives but it is recommended that you avoid complexity when building conditional code structures as it makes the code more difficult to maintain.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@if(anExpression)
...
someCode
...
@elif(anExpression)
...
someCode
...
@elif(anExpression)
...
someCode
...
@else
...
someOtherCode
...
@end @*/
```

See also:	Conditional code block, Pre-processing, Pre-processing - `/*@ ... @*/`

Pre-processing –@set (Pre-processor)

Set the contents of a pre-processor variable.

Availability:	JScript –3.0 Internet Explorer –4.0	
JavaScript syntax:	IE	`@set aVariable=aValue`
Argument list:	`aValue`	A value to assign
	`aVariable`	A pre-processor variable name

You can define variables that exist in the namespace of the pre-processor and which can then be used as if they were directives that you had created. For example, you might test some complex set of conditions and set a variable so that you can simply test for its existence later. Or you might use that variable in some fragment of code as a manifest constant, perhaps to specify the size of an array or a flag to activate some capability.

You should note that these pre-processor directives will likely not survive from one script block to another or for any duration in the time domain. They are not variables in the sense of a script variable.

To create a new pre-processor variable, use the `@set` directive, name the variable, and assign a value to it, like this:

```
@set @myvariable=1000
```

You can then use the variable in the source text like this:

```
document.write(@myvariable);
```

You may need to experiment to establish how long one of these variables actually persists. It is unlikely to still be defined when an event handler is called. However, that event handler may have been interpreted and stored when the script was loaded, in which case the variable would have been replaced by its value at that time.

If you intend to hide this directive inside some comments, it must be done like this:

```
/*@set @myvariable=1000 @*/
```

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

Precedence (Definition)

The logical order of evaluation of expressions according to predefined rules.

Availability:	ECMAScript edition –2
See also:	Associativity, Expression, Grouping operator ( ), Operator, Parentheses ( ), Operator Precedence

Cross-references:

ECMA 262 edition 2 –section –11.1.4

ECMA 262 edition 3 –section –11.1.6

Wrox *Instant JavaScript* –page –18

Preferences (Definition)

Browser preferences can sometimes be manipulated from JavaScript.

Netscape 4.0 introduced some new methods for storing preference settings in external JavaScript configuration files. Earlier version required an Admin Kit to be used to set defaults and the user could then only modify preferences from the options menu.

The JavaScript preferences information is contained in:

- `prefs.js`
- `netscape.cfg`
- `config.jsc`

The `prefs.js` file contains a collection of user preferences. It is read by the browser as it starts up and written by the browser as it shuts down. If you want to modify `prefs.js` by hand, you will need to do it while Netscape is not running. There are a few modifications that can be done to this file that will help when you are developing JavaScript source code.

Note that you can have a different `prefs.js` file for each user profile. On UNIX versions of Netscape Navigator, the `prefs.js` file becomes `preferences.js` instead.

Additional preference information may be stored in a file called `netscape.cfg`, which replaces the `netscape.lck` found on older versions of Netscape. This file overrides the settings in `prefs.js` but is not written to by the browser. It is an encrypted file that you cannot unwrap without a lot of work. It is possible to edit it carefully by hand but you really need the admin tools from Netscape to work on it. There is a tool called Mission Control that contains a configuration editor. There is also an install builder for delivering an encrypted file to the target system.

The third file is called `config.jsc` and overrides the other two. This file will likely only be present if you are running under the supervision of an administrator who uses the admin tools to configure hundreds of browsers. The `config.jsc` file can be loaded from a URL defined in `netscape.cfg`. Furthermore it can be read frequently and regularly by the browser so it is possible to turn features on and off during the day having configured `netscape.cfg` to re-read it from time to time.

Using these configuration scripts and the tools that Netscape provides, you can alter hundreds of parameter settings within the browser.

See also:	`.js,netscape.lck`

Cross-references:

Wrox *Instant JavaScript* –page –59

preferences.js (Special file)

A special Netscape Navigator file containing preference information.

See also:	`.js` Preferences

Cross-references:

Wrox *Instant JavaScript* –page –59

Prefix decrement (–) (Operator/prefix)

Decrement an operand before access.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`--anOperand`*
Argument list:	*`anOperand`*	A numeric value that can be decremented

The operand is evaluated, converted to a numeric value, and decremented by 1.

Although this is classified as a unary operator, its functionality is really that of an additive operator.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

See also:	Additive operator, Arithmetic operator, Associativity, Decrement value (--), Operator Precedence, Postfix decrement (--), Prefix expression, Prefix operator, Unary expression

Cross-references:

ECMA 262 edition 2 –section –11.4.5

ECMA 262 edition 2 –section –11.6.3

ECMA 262 edition 3 –section –11.4.5

Prefix expression (Operator/prefix)

Increment or decrement an operand before access.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

Prefix expressions operate on Left-Hand-Side (sometimes called LValue) expressions.

There are two prefix operators:

- ++ performs a numeric increment on the operand
- -- performs a numeric decrement on the operand

These can also be classified as additive operators and because they modify a value in place; they also imply that an assignment takes place as well.

See also:	Additive operator, Assignment operator, Decrement value (--), Expression, Increment value (++), Postfix decrement (--), Postfix expression, Postfix increment (++), Prefix decrement (--), Prefix increment (++)

Cross-references:

ECMA 262 edition 2 –section –11.3

ECMA 262 edition 3 –section –11.3

Prefix increment (++) (Operator/prefix)

Increment an operand before access.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server version –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*++anOperand*
Argument list:	*anOperand*	An incrementable numeric value

The operand is evaluated, converted to a numeric value, and incremented by 1.

Although this is classified as a unary operator, its functionality is really that of an additive operator.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

See also:	Additive operator, Arithmetic operator, Associativity, Decrement value (--), Operator Precedence, Prefix expression, Prefix operator, Unary expression

Cross-references:

ECMA 262 edition 2 –section –11.4.4

ECMA 262 edition 2 –section –11.6.3

ECMA 262 edition 3 –section –11.4.4

Prefix operator (Definition)

Operators that are placed before the operand.

Prefix operators are placed immediately before their operands. The following prefix operators are supported:

Operator	Description
++	Increment the operand
--	Decrement the operand

The meanings of some of these operators may vary in other contexts.

See also:	Arithmetic operator, Associativity, Operator, Operator Precedence, Postfix operator, Prefix decrement (--), Prefix increment (++), Unary operator

Cross-references:

Wrox *Instant JavaScript* –page –19

prefs.js (Special file)

A special Netscape Navigator file containing preference information.

See also:	.js, Preferences

Cross-references:

Wrox *Instant JavaScript* –page –59

Primary expression (Definition)

Primary expressions are used with operators to form more complex expression types.

Availability:	ECMAScript edition –2

A primary expression is one that needs no further evaluation to resolve its value.

A Primary expression is a specific object, identifier or literal and may also be the result of evaluating another nested expression that is surrounded by the grouping operators (parentheses).

The `this` keyword is classed as a Primary expression. The value returned by the `this` keyword depends on the execution context currently being processed.

An identifier is a primary expression if it refers to an object or function.

Constants and string literals are by definition primary expressions.

Any expression within the grouping operators (parentheses) becomes a primary expression since the rules of precedence dictate that it must be resolved completely before being replaced into the expression to which it is an operand.

An identifier is evaluated according to the current scoping rules presently in force. This would return an internal reference value inside the interpreter but this would be transparent to the user.

Literal values of the following types are considered to be primary expressions:

- Null literal
- Boolean literals
- Numeric literals
- String literals

Expressions can be evaluated in the desired order by using the grouping operator or parentheses to nest the expressions. This allows the `delete` and `typeof` operations to be applied to expressions.

See also:	Constant, Execution context, Expression, Grouping operator ( ), Identifier resolution, Literal, Operator Precedence, `this`

Cross-references:

ECMA 262 edition 2 –section –7.7

ECMA 262 edition 2 –section –10.1.4

ECMA 262 edition 2 –section –11.1

ECMA 262 edition 3 –section –10.1.4

ECMA 262 edition 3 –section –11.1

Primitive value (Definition)

A built-in native value type.

Availability:	ECMAScript edition –2

A primitive value is one of the types Undefined, Null, Boolean, Number, or String.

The foundation set of primitive values is represented at the lowest level of the language implementation within the core functionality sub-set. The host environment may add other primitive values.

Type Name	Description
Aggregate	A collection of atomic types assembled collectively into an object.
Arithmetic	All types that yield a value that can be operated on numerically.
Array	Collections of objects and identifiers assembled into a sequence.
Basic	The fundamental simple, non-object types.
Boolean	This type can store and yield `true` or `false` values.
Completion	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
List	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
Null	This has exactly one value, `null` and is distinct from `undefined`.
Number	Integer and floating-point values are all stored in a generic number type.
Object	An object is an unordered collection of properties. Each property consists of a name, a value, and a set of attributes.
Reference	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
Scalar	The non-object types.
String	Strings are arrays of characters that are accessible individually by indexing their position in the sequence.
Undefined	This value is returned by variables that have not yet been assigned a value.

Note that arrays and objects are not primitive types.

See also:	`Cast` operator, Character constant, Number, Number

Cross-references:

ECMA 262 edition 2 –section –4.3.2

ECMA 262 edition 3 –section –4.3.2

Wrox *Instant JavaScript* –page –14

print() (Method)

An alias for the `window.print()` method.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*`.print()`
	-	`print()`

Refer to:

`Window.print()`

Printing character (Definition)

A character with a visible glyph.

A printing character is one that given the current locale, represents a printable glyph or character shape.

The `isprint()` function (that is described elsewhere in this manual) lists printable characters.

See also:	Character handling, Control character, `isPrint()`, Letter

private (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

PrivilegeManager object (Java class)

A Java class that administers privileges.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myPrivilegeManager* = netscape.security.PrivilegeManager
Object methods:	disablePrivilege(), enablePrivilege()	

Because the Netscape security model is based on the Java security model, the Netscape Navigator browser requests its privileges through the Java mechanisms. These are encapsulated in a class that you can access from inside JavaScript.

The downside of this is that there is no meaningful value returned when the request is made. If the request for a privilege is denied, the error causes a Java exception that is difficult to trap from JavaScript. It is possible that more recent browser versions will support an exception-handling mechanism.

There are two principle methods that are useful here, one to request the privilege and the other to relinquish it.

- `enablePrivilege()` –Requests the privilege passed as a string argument
- `disablePrivilege()` –Relinquishes the privilege based on a string argument

It is good practice to disable the privilege as soon as you no longer need it. In any case the privilege is given up when the function that requested it exits.

Trying to examine an instance of this class leads to some interesting run-time errors. That is perhaps understandable since the object is involved with keeping things secret. Even after requesting privileges, you cannot examine the internals of an instance of this class.

See also:	`netscape.security.PrivilegeManager`, Requesting privileges, `UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`, `UniversalFileRead`, `UniversalPreferencesRead`, `UniversalPreferencesWrite`, `UniversalSendMail`

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
`disable Privilege()`	1.2 +	-	4.0 +	-	-	-	-	-	-	-	-
`enable Privilege()`	1.2 +	-	4.0 +	-	-	-	-	-	-	-	-

PrivilegeManager.disablePrivilege() (Method)

A method for removing a privilege from a user.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myPrivilegeManager*.disablePrivilege (*aPrivName*)
Argument list:	*aPrivName*	The name of a privilege to be relinquished

The method should be called with a string argument containing the name of the privilege that is no longer required.

Example code:

```
// Relinquish the file reading priviledge
netscape.security.PrivilegeManager.disablePrivilege("UniversalFileRead");
```

See also:	Requesting privileges, UniversalBrowserAccess, UniversalBrowserRead, UniversalBrowserWrite, UniversalFileRead, UniversalPreferencesRead, UniversalPreferencesWrite, UniversalSendMail

PrivilegeManager.enablePrivilege() (Method)

A method for granting an additional privilege to a user.

Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myPrivilegeManager*.enablePrivilege(*aPrivName*)
Argument list:	*aPrivName*	The name of a privilege to be requested

The method should be called with a string argument containing the name of the privilege that is needed to gain access to the secure facilities of the browser.

Example code:

```
// Request the file reading priviledge
netscape.security.PrivilegeManager.enablePrivilege("UniversalFileRead")
```

See also:	Requesting privileges, UniversalBrowserAccess, UniversalBrowserRead, UniversalBrowserWrite, UniversalFileRead, UniversalPreferencesRead, UniversalPreferencesWrite, UniversalSendMail

Privileges (Definition)

Secure access can be controlled by privileges.

Refer to:

Restricted access

Procedural surfaces (Definition)

A means of space filling an area within an HTML `Element` object using a shading algorithm.

This can be a means of making web page downloads much smaller because you specify an algorithm to use to fill a space rather than a pixel image map.

These are the available procedural surface generators you can use with MSIE:

- `AlphaImageLoader()`
- `Gradient()`

These procedural shaders compute the alpha channel transparency and the pixel color of the area in RGB coordinates.

See also:	filter – `alphaimageloader()`, filter – `gradient()`, `style.filter`, Visual filters

Procedure (Definition)

A procedure is a function that does not return a meaningful result and which is meant to be called outside of a context where its value will be assigned or substituted in an expression.

Refer to:

`function( ... ) ...`

ProcessingInstruction object (Object/DOM)

Part of the DOM level support for XML that relates to the handling of a processing instruction embedded in the text of the document.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0
Inherits from:	`Node` object
JavaScript syntax:	- *myProcessingInstruction* = new ProcessingInstruction()
Object properties:	`data, target`

See also:	`Document.createProcessingInstruction()`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	DOM	CSS	HTML	Notes
`data`	1.5 +	5.0 +	6.0 +	5.0 +	-	-	-	1 +	-	-	-
`target`	1.5 +	5.0 +	6.0 +	5.0 +	-	-	-	1 +	-	-	ReadOnly

Inheritance chain:

`Node` object

ProcessingInstruction.data (Property)

The data content of the processing instruction.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myProcessingInstruction`*`.data`

ProcessingInstruction.target (Property)

The target of the processing instruction is defined by XML as being the first token following the markup that begins the processing instruction.

Availability:	DOM level –1 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myProcessingInstruction`*`.target`

Property attributes:

`ReadOnly`.

Program (Definition)

Another name for a JavaScript script.

Refer to:

Script

project object (Object/NES)

A server-side host object provided inside NES. This object represents a running application inside the server.

Availability:	JavaScript –1.1 Netscape Enterprise Server version –2.0	
JavaScript syntax:	NES	`project`
Object methods:	`lock(), unlock()`	

The `client` object provides a means of maintaining state during a client session. This object is used for maintaining state across all sessions running in a single application. There may be several applications running in a server.

If you need to maintain state across the entire server then you need to access the `server` object which is discussed in a separate topic.

See also:	`client` object, Netscape Enterprise Server, `response.project`, `server` object, `unwatch()`, `watch()`

Method	JavaScript	JScript	NES	Notes
`lock()`	1.1 +	-	2.0 +	-
`unlock()`	1.1 +	-	2.0 +	-

project.lock() (Method)

A means of locking resources that might be shared by several sessions.

Availability:	JavaScript –1.1 Netscape Enterprise Server version –2.0	
JavaScript syntax:	NES	`project.lock()`

The project lock would be used at a higher hierarchical level than the `Lock` object that you might use for general purpose locking within a session.

It is even more important that you don't hog a lock on the `project` object as this can cause severe performance problems.

The lock will stall if another script currently has a lock extant on this project. The method will then return when that lock is relinquished.

If you are accessing files for writing in a server-side application (within NES), you should make sure the project locking is activated to avoid file corruption happening if there are multiple simultaneous accesses to the file.

Example code:

```
<SERVER>
// An example file access with project level locking to prevent
// file corruption.
project.lock();
myFileObject.open("a");
myFileObject.writeln("Append this line to the file.");
myFileObject.close();
project.unlock();
</SERVER>
```

See also:	`File.open(),server.lock()`

project.unlock() (Method)

Relinquish a lock on a project object.

Availability:	JavaScript –1.1 Netscape Enterprise Server version –2.0	
JavaScript syntax:	NES	`project.unlock()`

You should try and unlock any resources you have reserved with a lock as soon as you can.

See also:	`server.unlock()`

prompt() (Method)

An alias for the `window.prompt()` method.

Availability:	JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myResult* = *myWindow*.prompt(*aString*, *aDefaultValue*)
	-	*myResult* = prompt(*aString*, *aDefaultValue*)
Argument list:	*aDefaultValue*	An initial content for the text box
	aString	Some text to explain what to enter

See also:	Window.alert(),Window.confirm()

Refer to:

Window.prompt()

Property (Definition)

A property consists of a name, a value, and a set of attributes.

Availability:	ECMAScript edition –2

A property consists of a name, a value, and a set of attributes. It belongs to an object.

Since it belongs to an object, each instance of a particular object class can own its own private properties as well as inheriting shared properties from its prototype. Privately owned properties are sometimes called instance variables.

A property can have zero or more attributes. The attributes control how the property is accessed both internally from inside the interpreter and externally from your script.

A property is somewhat like a method, at least in the way it is described in a script. However, properties are containers whereas methods are actions. Properties can have values assigned to them or can have their values retrieved. A method will generally cause something within the receiving object to change. Properties are read-only, write-only, or read and write. Some properties are internal and private to the object and are not therefore exposed as scriptable items.

Methods are really references to function objects that can be called.

See also:	Function, Global object, java.util, Method, Object, Property accessor, Property attribute, Property name, Property value

Cross-references:

ECMA 262 edition 2 –section –15.1

ECMA 262 edition 3 –section –15.1

Property accessor (Definition)

Properties are accessed by name.

Availability:	ECMAScript edition –2

Properties are accessed by name either using the dot notation or the square bracket notation.

The dot and bracket notations are generally exchangeable with one another.

The object to which the property-access message is being directed can be derived from an expression.

The name of the identifier can be derived from an expression but it must yield a meaningful string value that corresponds to a genuine or potential property of the receiver.

See also:	Decimal point (.), Left-Hand-Side expression, Property

Cross-references:

ECMA 262 edition 2 –section –11.2.1

ECMA 262 edition 3 –section –11.2.1

Property attribute (Definition)

A property can have zero or more attributes.

Availability:	ECMAScript edition –2

A property can have zero or more attributes. The attributes control how the property is accessed both internally from inside the interpreter and externally from your script.

Here is a list of the property attributes defined by the ECMA Script standard:

Attribute	Description
`ReadOnly`	The property is a read-only value. Scripts will not be allowed to change the value although the value may change from time to time if it is dependant on some host related facility.
`DontEnum`	You cannot enumerate this property with a `for in` repetition.
`DontDelete`	You cannot delete this property.
`Internal`	This is an internal property, which you normally won't have any access to. It's likely it would be hidden inside the host-managed objects and inaccessible to your scripts.

Where properties are `ReadOnly`, this is flagged in the documentation. Read/Write access is assumed to be the default case otherwise.

There is conflicting information in the reference sources regarding the read/write ability of some properties. Some suggest a particular property is `ReadOnly` and others suggest you can assign a value to it. It may be that you can assign a value to it without the JavaScript interpreter complaining but that any value you assign is ignored.

See also:	delete, Object, Property

Cross-references:

ECMA 262 edition 2 –section –8.6.1

ECMA 262 edition 3 –section –8.6.1

Property name (Definition)

The name of an object property.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0

Accessing properties of an object by name simply requires the name to be added to the object reference with a dot separator between them.

However, the properties in an object are also kept in an array. You can access items in an array by index number. However, the objects are really accessed by name and so item 4 is really accessed as item "4". You can use the property names as symbolic names for an array index.

Now, assuming that we had deduced that the property we were interested in was stored in item 4 of the property array and had the name `property4` as well, all of these would access the same property:

- `myObject.4`
- `myObject.property4`
- `myObject[4]`
- `myObject["4"]`
- `myObject["property4"]`

See also:	Array index delimiter ([ ]), Property

Cross-references:

ECMA 262 edition 2 –section –8.6.1

ECMA 262 edition 3 –section –8.6.1

Wrox *Instant JavaScript* –page –32

Property value (Definition)

The value of an object property that is returned when that property is requested.

Availability:	ECMAScript edition –2

Objects own a set of properties that are named containers for values. You can create new properties belonging to the object itself or to its prototype if you want to share them among all instances of an object class. Using the object and property reference as an LValue allows you to assign a new value to the property. To retrieve the property value, use it as you would any other RValue.

See also:	LValue, Property, RValue

Cross-references:

ECMA 262 edition 2 –section –8.6.1

ECMA 262 edition 3 –section –8.6.1

protected (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Prototype–Based Inheritance (Definition)

JavaScript supports an inheritance chain based on prototypes.

Availability:	ECMAScript edition –2

JavaScript supports an inheritance chain based on prototypes. Every constructor has an associated prototype and every object created with that constructor has a link to the prototype as an integral part of its instantiation. This is called the Object's Prototype.

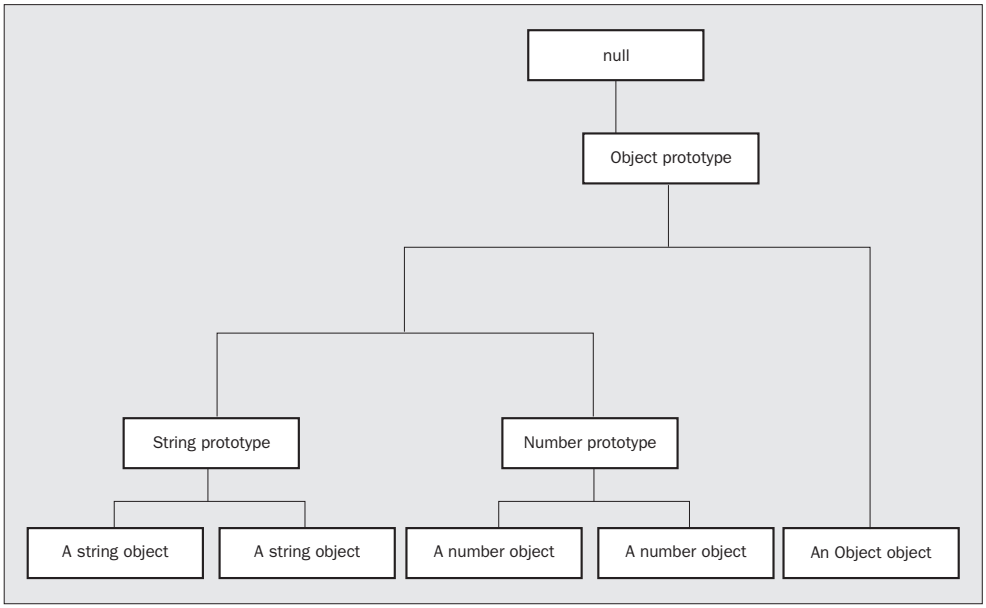
A prototype may have an implicit reference to it's parent prototype. This provides inheritance through a Prototype Chain that is analogous to the Super-Class and Sub-Class mechanisms in a class-based object-oriented language.

This allows for properties to be overridden or provided by parent prototypes if the properties are not implemented in the target object.

In a class-based object-oriented environment, as a general rule, state values are embodied in object instances of a class but methods are contained in the classes themselves so the inheritance only projects structure and behavior down through the sub-classing mechanism.

In the prototype-based inheritance the state AND methods are carried by the objects so structure, state, and behavior are all inherited down the prototype chain.

All objects that do not contain a particular property and that are descended via the inheritance tree from a single object that does contain that property will all share that one single property instance.



See also:	`function( ... ) ...`, Hierarchy of objects, Inheritance, Namespace, Prototype chain, `prototype` property, Shared Property

Cross-references:

ECMA 262 edition 2 –section –4.2.1

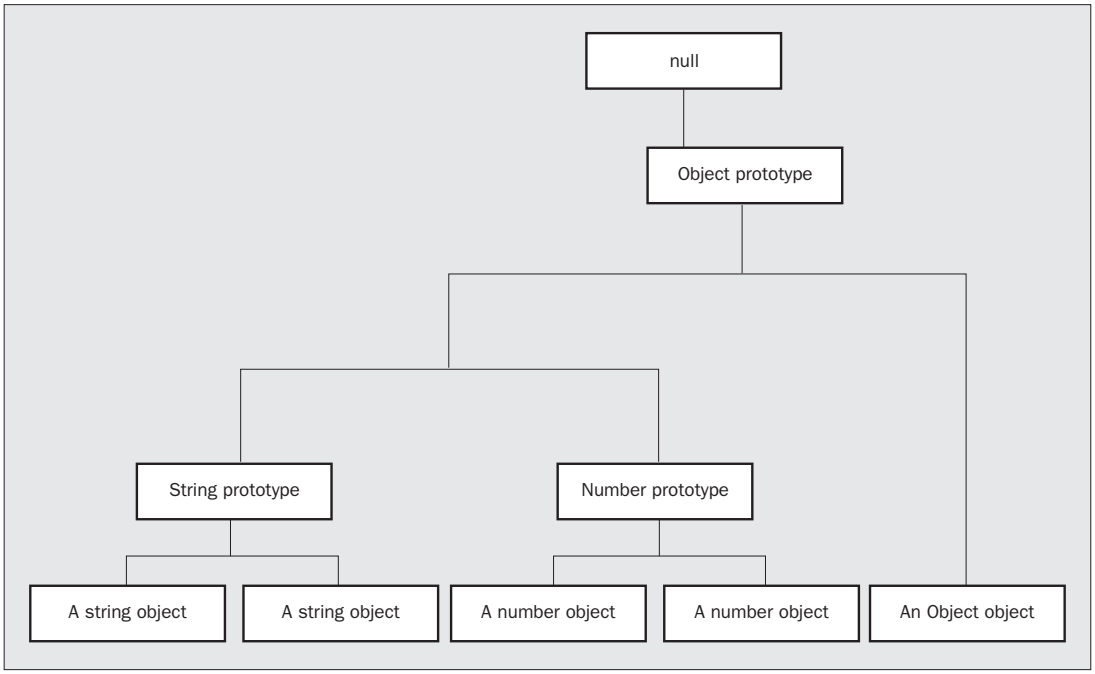
ECMA 262 edition 3 –section –4.2.1

Prototype chain (Definition)

JavaScript supports an inheritance chain based on prototypes.

Availability:	ECMAScript edition –2

A prototype chain is analogous to the super/sub-class inheritance mechanisms in an object-oriented environment. It is not quite the same, however, because the inheritance takes place at the object level and there are no true classes to instantiate. Instead you clone existing objects.



See also:	Prototype Based Inheritance, `prototype` property, Shared Property

Cross-references:

ECMA 262 edition 2 –section –4.2.1

ECMA 262 edition 2 –section –8.6.2

ECMA 262 edition 3 –section –4.2.1

ECMA 262 edition 3 –section –8.6.2

Prototype object (Definition)

Prototypes are analogous to default instances in a truly object-oriented system.

Availability:	ECMAScript edition –2

Prototypes are analogous to default instances in a truly object-oriented system. A constructor would copy this default instance to create a new instance of its object class.

A prototype is an object that implements structure, state, and behavior inheritance. When a constructor creates a new object, that new object implicitly refers to the constructor's associated prototype to resolve property references.

Prototypes can be referenced using the dot separated object hierarchy notation. With an object constructor called `myObject`, its prototype would be accessed like this:

```
myObject.prototype
```

If any properties are added to the prototype, they will be shared by and available to all objects created by the constructor associated with that prototype. Such objects may override the inheritance by having identically named properties added to them directly.

A Prototype would be expected to support the following property by default:

- `constructor`

It should also support the following method by default:

- `toString()`

Cross-references:

ECMA 262 edition 2 –section –4.3.5

ECMA 262 edition 3 –section –4.3.5

prototype.constructor (Definition)

The prototype object has a constructor that refers to the object that the prototype object is a property of.

See also:	Shared Property Constructor property

prototype.toString() (Definition)

A method that you should redefine in your own classes to yield a meaningful string value.

When you create a custom object type of your own, you add various methods to it to give it some capabilities that are unique to that type (otherwise why are you creating a new object type?).

Unless you override the default `toString()` method will be provided by the object that you cloned to make your class. By default that will be the global `Object` object.

The example show how to do this in a way that is consistent with normal JavaScript behavior.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>

// Make a constructor for a new class
function Car()
{
   this.myClass = "Car";
}

// Create a special toString() override
function myToString()
{
   return "[object " + this.myClass + "]";
}

// Register our overriding toString()
Car.prototype.toString = myToString;

// Instantiate an object from the new class
var myCar = new Car();

// Test the new object
document.write("To string : " + myCar.toString() + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`toString()`

prototype property (Definition)

An internal method that returns a prototype.

Availability:	ECMAScript edition –2
Property/method value type:	Depends on the object

This property returns the prototype for the containing object.

The `prototype` property for an object returns a reference to the object that is the parent of the receiver in the prototype inheritance chain.

Any methods, properties, or functions present in the prototype will also be available in the child unless they are overridden.

This means that you can locate the parent prototype for a group of objects and store a value into a property there and it will be immediately available as a property of all the objects that share the same prototype.

The typical time you do this is when you are creating a prototype for a new class of object of your own. In that case you would also create a constructor function to initialize new instances of the object as well.

This property will either return `null` or an object that is the prototype for the object responding to the property request. This is used for maintaining the inheritance through the prototype chain.

Properties of the prototype object are exposed to the child object through the get accessor method. However since they are shared, they cannot be changed with the put accessor through the child. If they are changeable at all, the specific object that explicitly owns them is the only one with rights to modify the values as long as they are not read-only. This prevents a child object from modifying a property and that change propagating across all the objects that share the same property between them.

There is an internal property called `Prototype`, which yields the same value. Note the capitalization. Internal properties are not generally exposed to the scripting language. The `Prototype` property is one of the few that are.

Prototypes are also useful as a means of extending the interface for a particular type of object. By adding new properties and methods to the prototype, you make them available to all objects of that class.

Warnings:

- Do not create your own object properties with this name. You can assign new values to the prototype property in some implementations if you want to modify the inheritance chain but you should not use this property name for any other purpose.
- You cannot assign values to the prototype of the native objects in MSIE version 3.
- Be careful if you are adding to the prototype for some of the more obscure classes as this may reveal some shortcomings in the prototype-constructor mapping, which is implemented incorrectly for some objects.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Make a constructor for a new class
function Car(engineSize)
{
   this.enginesize   = engineSize;
   this.totalMileage = 0;
}

// Create a special toString() override
function myToString()
{
   return "This is a car object";
}

// Register our overriding toString()
Car.prototype.toString = myToString;
```

```
// Make a couple of other methods to add to it
function mileage()
{
   return this.totalMileage;
}
function addMiles(newMiles)
{
   this.totalMileage += newMiles;
}

// Add some shared properties
Car.prototype.wheels = "Alloy";
Car.prototype.body   = "Cabriolet";

// Register the mileage methods
Car.prototype.mileage  = mileage;
Car.prototype.addMiles = addMiles;

// Instantiate an object from the new class
var myCar = new Car(2000);

// Drive it around some to add miles to its clock
myCar.addMiles(100);
myCar.addMiles(150);
myCar.addMiles(200);

// Display its properties and call its methods
document.write("To string : " + myCar.toString() + "<BR>");
document.write("Engine size : " + myCar.enginesize + "<BR>");
document.write("Wheels : " + myCar.wheels     + "<BR>");
document.write("Body : " + myCar.body       + "<BR>");
document.write("Mileage : " + myCar.mileage()  + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Array.prototype`, `Boolean.prototype`, Constructor function, `constructor` property, `Date.prototype`, `Error.prototype`, `Function.prototype`, `Global` object, Internal Property, Native object, `Number.prototype`, `Object.prototype`, `Option.prototype`, Prototype-Based Inheritance, Prototype chain, `RegExp.prototype`, Shared Property, `String.prototype`

Property attributes:

`DontEnum`, `Internal`.

Cross-references:

ECMA 262 edition 2 –section –8.6.2

ECMA 262 edition 3 –section –8.6.2

Wrox *Instant JavaScript* –page –31

Proxies (Definition)

A proxy server mediates on your behalf to span a firewall and gather items that your browser has requested.

This won't generally have any impact on the coding of your JavaScript but in the case of the Netscape Navigator browser, the proxy connection setup can be defined in a `proxy.pac` file that is coded using JavaScript to describe the proxy preference settings.

See also:	`.pac,FindProxyForURL(),isInNet(),proxy.pac`

proxy.pac (Special file)

This is a special file containing rules for accessing sites through proxy servers.

If you are using a browser inside a firewall, you will likely reach the internet by using a proxy server. This is a special kind of web server that spans or bridges the firewall and fetches things across the internet for you.

Generally these proxy servers will also limit access or log details of everything you fetch.

You can operate with no proxies, manually defined proxies, or automatic proxies in the case of Netscape Navigator.

When you select Automatic mode, you are connecting in to a piece of JavaScript code that can work out whether to use a proxy or not and if so, which one.

In this context, some language features may be unavailable but then others are provided to assist in the deconstruction of URL values, and are only available in this context.

The only purpose of a `proxy.pac` file is to define the content of the `FindProxyForURL()` function.

The `proxy.pac` file can be retrieved from any location that can be defined by a URL. This means it could be a local file on your desktop or served directly by a web server inside your firewall. In theory it could be served by a web server outside your firewall so long as your firewall had a 'hole' in it to allow you to gain direct access to the server. That could lead to problems where the file you pull back might thereafter prevent access to that location, and it obviates the whole purpose of having a firewall in the first place.

Providing this file on a local web server inside your firewall means that it can be shared by all your Netscape Navigator browser users and maintained from a central locations. This is not very much use if you only have one user but when you have 500 it is a great time saver. However, the downside is that if you publish a broken `proxy.pac` file, all of your users go offline as soon as their browsers download it.

You cannot browse a `proxy.pac` file with a Netscape Navigator browser; however, you might be able to download one with MSIE if you are curious to see what it looks like.

To set this mechanism working, you need to go to the proxy configuration panel in your browser preferences and choose automatic proxy configuration. Then you need to type in the URL where the `proxy.pac` file lives.

See also:	.pac, File extensions, FindProxyForURL(), isInNet(), isPlainHostName(), Proxies

Cross-references:

Wrox *Instant JavaScript* –page –57

Pseudo-random numbers (Definition)

A series of numbers having an apparently random distribution.

A random number is generated from a genuinely unpredictable and non repeating sequence.

A pseudo-random number is generated from a series of numbers having a distribution which may not be truly random, but which is sufficiently complex and/or nonlinear that it is not readily distinguishable from a random sequence.

See also:	Math.random()

public (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 –section –7.4.3

ECMA 262 edition 3 –section –7.5.3

Punctuator (Definition)

Punctuators are composed of special non-alphabetic characters.

Availability:	ECMAScript edition –2

Punctuators are composed of special non alphabetic characters and are considered to be valid tokens by the interpreter.

Here is a summary of all the valid tokens:

Value	Meaning
!	Logical not
!=	Not equal to
!==	Not exactly equal to (ECMA edition 3)
%	Remainder
%=	Remainder and assign
&	Bitwise AND
&&	Logical AND
&=	Bitwise AND and assign
(	Function argument delimiter and precedence control
)	Function argument delimiter and precedence control
*	Multiply
*=	Multiply and assign
+	Unary plus, add, concatenate string
++	Postfix and prefix increment
+=	Add and assign
,	Argument delimiter
-	Unary minus, subtract
--	Postfix and prefix decrement
-=	Subtract and assign
.	Property accessor
/	Divide
/=	Divide and assign
:	Part of conditional operator
;	Statement terminator
<	Less than
<<	Bitwise shift left
<<=	Bitwise shift left and assign
<=	Less than or equal to
=	Assign value
==	Equal to
===	Exactly equal to (ECMA edition 3)
>	Greater than
>=	Greater than or equal to
>>	Bitwise shift right
>>=	Bitwise shift right and assign

Table continued on following page

Value	Meaning
>>>	Bitwise shift right (unsigned)
>>>=	Bitwise shift right (unsigned) and assign
?	Conditional operator
[	Array index delimiter
]	Array index delimiter
^	Bitwise XOR
^=	Bitwise XOR and assign
{	Start code block
|	Bitwise OR
|=	Bitwise OR and assign
||	Logical OR
}	End code block
~	Bitwise NOT

Refer to the items in the see-also list for more details of their functionality.

The [ ], { }, and ( ) punctuators must be used in pairs and must also be nested correctly.

Some punctuator symbols are also used as operators.

See also:	Code block delimiter {}, Lexical element, Operator, Operator Precedence, Statement, Token

Cross-references:

ECMA 262 edition 2 –section –7.6

ECMA 262 edition 3 –section –7.7

Put() (Function/internal)

Internal private function.

Availability:	ECMAScript edition –2

This internal function is used to store a new value into an internal property.

The property is set or not according to the result of the `CanPut` function for the property. If the property does not exist, then a new property is created with its attributes set to an empty condition.

The `Put` internal function may indeed allow a property value to be changed when received by a host object, even if that host object would respond to the `HasProperty` function with a `false` result indicating that the property does not exist.

See also:	Internal Method, `PutValue()`

Property attributes:

`Internal`.

Cross-references:

ECMA 262 edition 2 –section –8.6.2.2

ECMA 262 edition 3 –section –8.6.2.2

put() (Method/internal)

Write publicly accessible properties.

Availability:	ECMAScript edition –2

Used to store new values into publicly exposed properties.

See also:	Accessor method

Cross-references:

ECMA 262 edition 2 –section –8.6.2.2

ECMA 262 edition 3 –section –8.6.2.2

PutValue() (Function/internal)

Internal private function.

Availability:	ECMAScript edition –2

This internal function stores a value in the property belonging to the reference item passed as its argument.

A run-time error is generated if the argument passed in is not a reference item.

If the target object does exist, the usual `Put()` function logic is invoked to store the value in the object property.

If the target object does not exist, then the property is added to the global object and takes the value that is passed as a new value.

See also:	`Global` object, `Put()`, Reference

Property attributes:

`Internal`.

Cross-references:

ECMA 262 edition 2 –section –8.7.4

ECMA 262 edition 3 –section –8.7.2

Q object (Object/HTML)

An object instantiated by the HTML `<Q>` tag which indicates that a part of the document content is a quotation.

Availability:	JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myQ* = *myDocument*.all.*anElementID*
	IE	*myQ* = *myDocument*.all.tags("Q")[*anIndex*]
	IE	*myQ* = *myDocument*.all[*aName*]
	-	*myQ* = *myDocument*.getElementById(*anElementID*)
	-	*myQ* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myQ* = *myDocument*.getElementsByTagName("Q")[*anIndex*]
HTML syntax:	`<Q> ... </Q>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	An `ID` attribute value
Object properties:	`cite`	
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`cite`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

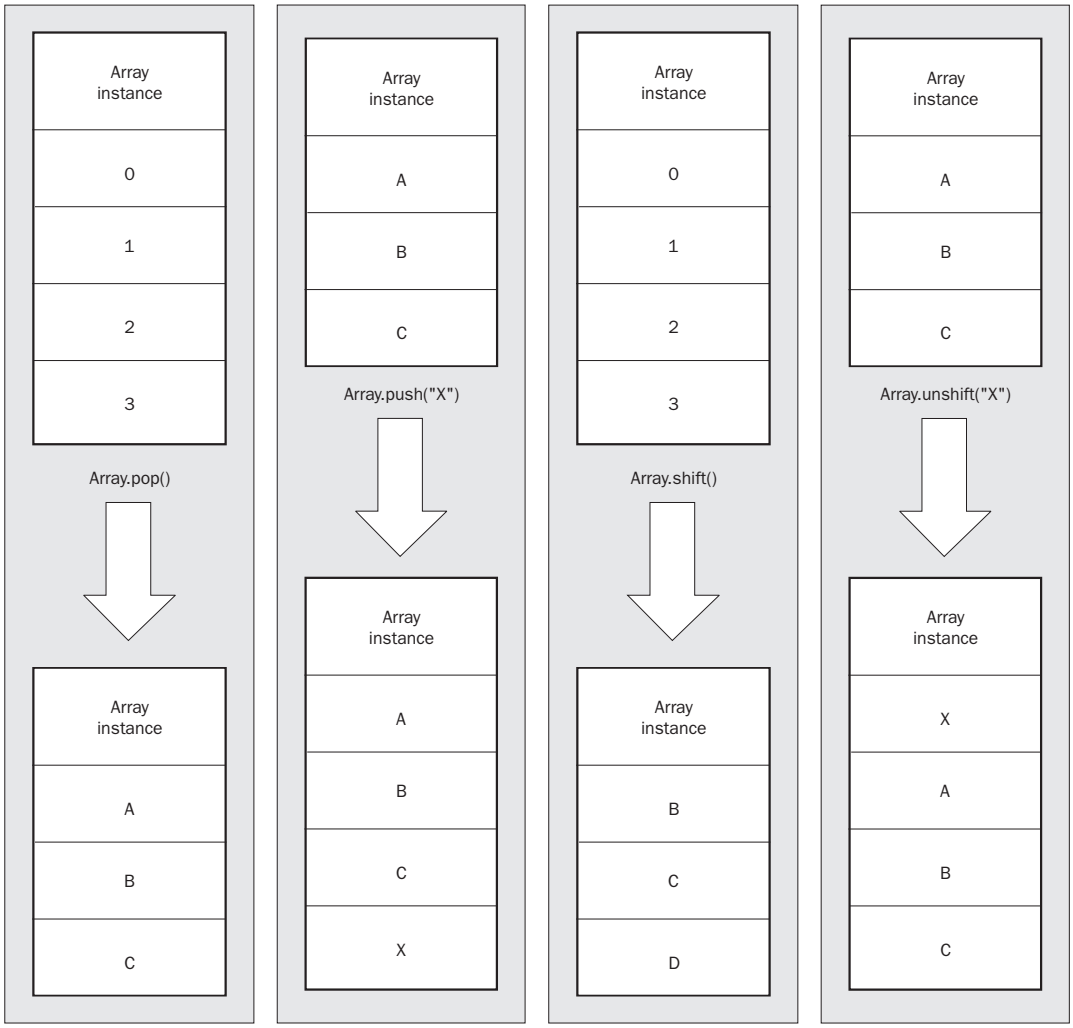
Queue manipulation (Useful tip)

With the enhanced array manipulation tools of JavaScript, you can build queue managers.

With the `Array` object methods `push()`, `pop()`, `unshift()` and `shift()`, you can build various stacks and queues.

'First In First Out' queues can be built with either `unshift()` and `pop()` or `push()` and `shift()` depending on which direction you want the objects to be queued in.

These capabilities have been available in Netscape since version 4, but are only available in MSIE at version 5.5, having been added to gain ECMA edition 3 compliance.



See also:	`Array.pop()`,`Array.push()`,`Array.shift()`,`Array.unshift()`, Stack manipulation

Quotation mark (" and ') (Delimiter)

String literal delimiter.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.0 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`'ABCDEF'`
	-	`"ABCDEF"`

See also:	ASCII, Escape sequence (\), Escaped JavaScript quotes in HTML, Line terminator, Literal, Punctuator, String, Unicode, `var`, String literal

Cross-references:

ECMA 262 edition 2 –section –7.7.3

ECMA 262 edition 2 –section –7.7.4

ECMA 262 edition 3 –section –7.8.4

R.E. (Definition)

Another name for a regular expression.

Refer to:

Regular expression

RadialWipe() (Filter/transition)

A transition effect similar to that seen on radar displays.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `RadialWipe()`

RadioButton object (Object/DOM)

A toggle button that acts together with a group of radio buttons in a family. Clicking one deselects any others in the group. These are used in forms to choose one item from a set.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0
Inherits from:	`Input` object

JavaScript syntax:	-	*myRadioButton* = *myDocument*.*aFormName*.*anElementName*
	-	*myRadioButton* = *myDocument*.*aFormName*.elements[*anItemIndex*]
	IE	*myRadioButton* = *myDocument*.all.*anElementID*
	IE	*myRadioButton* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*myRadioButton* = *myDocument*.all[*aName*]
	-	*myRadioButton* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*myRadioButton* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*]
	-	*myRadioButton* = *myDocument*.getElementById(*anElementID*)
	-	*myRadioButton* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myRadioButton* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
HTML syntax:	<INPUT TYPE="radio">	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The name attribute of an element
	anElementID	The ID attribute of an element
	anItemIndex	A valid reference to an item in the collection
	aFormIndex	A reference to a particular form in the forms collection
Object properties:	checked, defaultChecked, status, type, value	
Object methods:	handleEvent()	
Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onClick, onDblClick, onErrorUpdate, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onRowEnter, onRowExit	

Many properties, methods, and event handlers are inherited from the Input object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the Input object super-class.

There isn't really a RadioButton object class but it is helpful when trying to understand the wide variety of input element types if we can reduce the complexity by discussing only the properties and methods of a radio button. In actual fact, the object is represented as an item of the Input object class.

The RadioButton sub-class of the Input object does not support the select() method or the defaultValue property except on MSIE.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="RESULT">?</DIV>
<FORM onClick="handleClick()">
<INPUT TYPE="radio" VALUE="A" NAME="SET">Selection A<BR>
<INPUT TYPE="radio" VALUE="B" NAME="SET">Selection B<BR>
<INPUT TYPE="radio" VALUE="C" NAME="SET">Selection C<BR>
<INPUT TYPE="radio" VALUE="D" NAME="SET">Selection D<BR>
</FORM>
<SCRIPT>
//Code for IE only
function handleClick()
{
   myString  = "[";
   myString += document.forms[0].elements[0].checked;
   myString += "] [";
   myString += document.forms[0].elements[1].checked;
   myString += "] [";
   myString += document.forms[0].elements[2].checked;
   myString += "] [";
   myString += document.forms[0].elements[3].checked;
   myString += "]";
   document.all.RESULT.innerText = myString;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Element object, `Form.elements[]`, `FormElement` object, `Input` object, `Input.accessKey`, `onClick`, `RadioButton.handleEvent()`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`checked`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`defaultChecked`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`status`	-	3.0 +	-	4.0 +	-	-	-	-
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onClick	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onErrorUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

RadioButton.checked (Property)

The state of the button is returned by this property.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`myRadioButton`*`.checked`
HTML syntax:	`<INPUT CHECKED>`	

If the radio button has a mark in it (it is the only one of the family it belongs to that is selected), then this value will return `true`. Otherwise it will return `false`.

RadioButton.defaultChecked (Property)

The default checked state for a radio button in a form.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *`myRadioButton`*`.defaultChecked`
HTML syntax:	`<INPUT CHECKED>`

The `defaultChecked` state of an `Input` item is the value that was defined in the HTML document source when the page was loaded. You can use this value if you need to reset the status of a page or determine whether the user changed the settings on an input item since the page was loaded.

RadioButton.handleEvent() (Method)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N *`myRadioButton`*`.handleEvent(`*`anEvent`*`)`	
Argument list:	*`anEvent`*	An event to be handled by this object

This applies to Netscape Navigator prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`handleEvent()`, `RadioButton` object

RadioButton.status (Property)

The current status of a particular radio button.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myRadioButton`*`.status`

This is the current status of the radio button item. It is either checked or not. If the radio button has not been changed since the page was loaded from the server, then this value will be the same as the `defaultChecked` state of the radio button.

RadioButton.type (Property)

The type value for the `<INPUT>` object that describes the radio button.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myRadioButton`*`.type`

The type value for a `RadioButton` is always "radio". This value is necessary to determine the type of form element because this object is really an instance of the `Input` class and not a `RadioButton` class, because there is no `RadioButton` class.

See also:	`Input.type`

Property attributes:

`ReadOnly`.

RadioButton.value (Property)

The text string value of this particular radio button.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myRadioButton`*`.value`

This value is returned to the server during a submit only if the checked state for this radio button is on.

Warnings:

- The state of the checkbox is in the checked property not the value property.

See also:	`Input.value`

RandomBars() (Filter/transition)

A transition effect with the appearance of random bars sliding down.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `RandomBars()`

RandomDissolve() (Filter/transition)

A transition effect with the appearance of a fine pixelated dissolve.

Availability:	JScript –5.5 Internet Explorer –5.5

Refer to:

Filter – `RandomDissolve()`

Range error (Definition)

Some functions (especially math functions) have a limited range of valid arguments.

Range errors occur when the value passed to a function falls outside the range of possible valid values for that function. This problem is most prevalent with functions belonging to the `Math` object.

An out-of-range error is caused because the function cannot resolve the input value to a meaningful output value.

Because JavaScript is more forgiving than a compiled language, these exceptions are managed by returning one of the following values:

- `NaN`
- `+Infinity`
- `-Infinity`
- `undefined`
- `null`

The host implementation may provide other values and, if it so chooses, can deliver a specific range error value for its own function calls.

Refer to descriptions of the `Math` object and its functions for details of the range of suitable values for each function call.

See also:	`Infinity`, `Math` object, Mathematics, Minima-maxima, `NaN`, Null literal, undefined, Undefined behavior

RangeError object (Object/core)

A native error object based on the `Error` object.

Availability:	ECMAScript edition –3 JavaScript –1.5 Netscape –6.0	
Inherits from:	&`Error object`	
JavaScript syntax:	N	*`myError`* `= new RangeError()`
	N	*`myError`* `= new RangeError(`*`aNumber`*`)`
	N	*`myError`* `= new RangeError(`*`aNumber, aText`*`)`
Argument list:	*`aNumber`*	An error number
	`aText`	A text describing the error

This sub-class of the `Error` object is used when an exception is caused by a numeric value exceeding its allowable range.

See also:	`catch( ... )`, `Error` object, `EvalError` object, `ReferenceError` object, `SyntaxError` object, `throw`, `try ... catch ... finally`, `TypeError` object, `URIError` object

Inheritance chain:

`Error` object

Cross-references:

ECMA 262 edition 3 –section –15.1.4.11

ECMA 262 edition 3 –section –15.11.6.2

Raw event (Definition)

An event that describes a physical action.

See also:	`onBlur`, `onFocus`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseDrag`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, Event propagation

ReadOnly (Property attribute)

An internal property attribute that controls whether a property value can be changed.

Availability:	ECMAScript edition –2
Property/method value type:	`Boolean primitive`

This is intended to prevent the script from modifying a property value.

You should note however that this does not mean that the property must always be `ReadOnly`. There may be times when the hosting environment changes the `ReadOnly` settings for a property.

Generally, the `ReadOnly` state would not change very often but the standard mandates that it does not mean constant and unchanging, only that while the `ReadOnly` state is `true`, the property should be locked out against any attempts to change its value.

Note that this is not the `readOnly` property that belongs to a Form Input element. That is intended to stop the user from altering the current state of an input item.

See also:	`DontDelete`, `DontEnumerate`, `Input.readOnly`

Cross-references:

ECMA 262 edition 2 –section –8.6.1

ECMA 262 edition 3 –section –8.6.1

Rect object (Object/browser)

A rectangle object used for Layer clip rectangles.

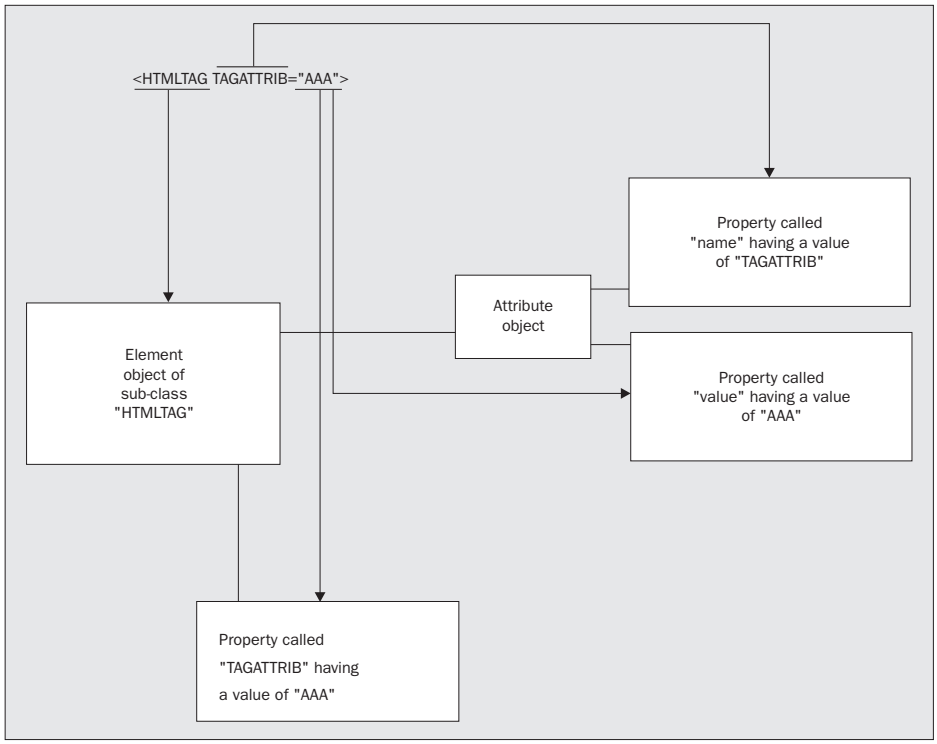
Availability:	JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	*myRect* = myLayer.clip
	N	*myRect* = myStyle.clip
Object properties:	bottom, height, left, right, top, width	

This object represents a clipping rectangle that the visible part of a display object is viewed through. This is most likely used with a `layer` object. The layer contents would be drawn off-screen and then that part which falls within the clipping rectangle would be displayed in the window.

This can be useful for performing wipes and making parts of a layer progressively visible within some kind of transition loop.

In the MSIE browser, these rectangular objects are manufactured as needed with the `rect()` constructor function.

These rectangles are not the same as you can create with the `getBoundingClientRect()` method, which applies to a `TextRange` object. That method creates `TextRectangle` which responds differently to the pixel rectangle we have here.



See also:	`Clip` object, `Layer.clip`, `style.clip`, `TextRange.getBoundingClientRect()`, `textRectangle` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`bottom`	1.2 +	-	4.0 +	-	-	-	-
`height`	1.2 +	-	4.0 +	-	-	-	-
`left`	1.2 +	-	4.0 +	-	-	-	-
`right`	1.2 +	-	4.0 +	-	-	-	-
`top`	1.2 +	-	4.0 +	-	-	-	-
`width`	1.2 +	-	4.0 +	-	-	-	-

Rect.bottom (Property)

The bottom edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myRect*`.bottom`

This defines the bottom edge of the rectangle described by the object. You could modify this in a loop to create a vertical upwards wipe transition effect.

See also:	`Clip.bottom`, `Layer.clip.bottom`

Rect.height (Property)

The height of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myRect*`.height`

The clip region is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

See also:	`Clip.height`, `Layer.clip.height`

Rect.left (Property)

The left edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myRect*`.left`

This defines the left edge of the clip region. You could modify this in a loop to create a horizontal wipe transition effect.

See also:	`Clip.left,Layer.clip.left`

Rect.right (Property)

The right edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myRect*`.right`

This defines the right edge of the clip region. You could modify this in a loop to create a horizontal wipe transition effect.

See also:	`Clip.right,Layer.clip.right`

Rect.top (Property)

The top edge of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myRect*`.top`

This defines the top edge of the clip region. You could modify this in a loop to create a vertical downwards wipe transition effect.

See also:	`Clip.top,Layer.clip.top`

Rect.width (Property)

The width of a layer's clip region.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	`myRect.width`

The clip region is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

See also:	`Clip.width`, `Layer.clip.width`

Reference (Definition)

An internal type used by the interpreter.

Availability:	ECMAScript edition –2

This is an internal type used by the interpreter for processing expression evaluation results. It cannot be stored as an object property.

JavaScript supports references to objects via variables that contain them. This is a useful facility to be able to make use of as it can increase performance and also allow code to be reused.

Conceptually at least, the storage of objects in variables works as if the variable assumed a `Reference` type.

A `Reference` item points at the property of an object. Therefore it consists of two parts: the base object and the property name.

It is ephemeral and is not necessarily implemented in the interpreter, but it is used in the ECMA standard to help explain how some of the internal algorithms operate.

It facilitates a simplified syntax grammar and therefore adds value in the understanding of how the internals of the grammar should behave.

It helps describe the operation of the `delete` and `typeof` operators, the assignment of values and function calls. It also assists in the grammar of the '`this`' value for function calls that are associated with objects and used as methods.

See also:	Assign value (=), Function arguments, `GetBase()`, `GetPropertyName()`, `GetValue()`, `PutValue()`, `this`, Type, `typeof`

Cross-references:

ECMA 262 edition 2 –section –8.7

ECMA 262 edition 3 –section –8.7

Reference counting (Definition)

The manner in which we keep track of object usage.

Using the `new` operator creates a new object with a reference count of zero. Assigning that object creation to a variable increments the reference count as does storing it in an array or saving it as an object property. Deleting a variable containing a reference to an object decrements the reference count for that object.

When you assign the result of a new operation to a variable, first the object gets instantiated and stored and a reference to it is stored in the variable. If you assign the variable to another, you copy the reference. Now two variables are referring to the same object. Modifying the object that one points at modifies the other.

Using the `delete` operator on one variable removes a reference to the object. The other variable continues to refer to it and the object must remain persistent while it does. When no variables are referring to the object and there are no other references such as items in arrays or properties in other objects, the reference count is zero.

When the reference count is zero, the object can be garbage collected. Also, because there are no references to it, you have no handle by which you can reach it, so if it isn't garbage collected, it will waste the space it occupies. When that happens, you have a memory leak.

You would not normally have any access to the reference count for a variable and indeed, some implementations may use a different technique to manage garbage collection. However, it is a useful concept and allows the idea of objects, references and garbage collection be more easily understood.

See also:	`delete`, Function arguments, Garbage collection, Memory leak, `Object()`, `Option()`, Variable

ReferenceError object (Object/core)

A native error object based on the `Error` object.

Availability:	ECMAScript edition –3 JavaScript –1.5 Netscape –6.0	
Inherits from:	`Error` object	
JavaScript syntax:	N	*myError* `= new ReferenceError()`
	N	*myError* `= new ReferenceError(`*aNumber*`)`
	N	*myError* `= new ReferenceError(`*aNumber*`, `*aText*`)`
Argument list:	*aNumber*	An error number
	aText	Text describing the error

This sub-class of the `Error` object is used when an exception is caused by an incorrect reference being made to an object.

See also:	catch(...), `Error` object, `EvalError` object, `RangeError` object, `SyntaxError` object, `throw`, `try` ... `catch` ... `finally`, `TypeError` object, `URIError` object

Inheritance chain:

`Error` object

Cross-references:

ECMA 262 edition 3 –section –15.1.4.12

ECMA 262 edition 3 –section –15.11.6.3

Regex (Definition)

Another name for regular expression handling.

See also:	`RegExp` pattern –character literal, Regular expression

RegExp literal (Definition)

A way of creating and initializing regular expression objects.

Availability:	ECMAScript edition –3

A `RegExp` literal is defined as some matching expression enclosed in slash characters.

The grammar for building regular expressions is somewhat complex. The rules are basically straightforward with there being a sequence of individual matching rules assembled together and enclosed in slashes. There can be some modifier flags placed after the second (terminating) slash delimiter.

Here are some example Regular Expression literals:

```
/^JavaScript/

/19[0-9][0-9]*/

/\binterpreter/i

/squeek/g
```

You can assign these expressions to variables, which will then contain a `RegExp` object.

Warnings:

- You cannot define a `RegExp` literal that is empty simply with a pair of slash characters. This would define a single line comment delimiter instead.
- You must specify an empty `RegExp` literal like this:
  - `/(?:)/`

Example code:

```
// Declare a variable and assign a RegExp literal to it
var myPattern = /x$/;
// Now do the same thing with a RegExp constructor
var myPattern = new RegExp("x$");
```

See also:	`RegExp` pattern –character literal, `RegExp()`

Cross-references:

ECMA 262 edition 3 –section –7.8.5

O'Reilly *JavaScript Definitive Guide* –page –49

RegExp object (Object/core)

An object that encapsulates regular expressions.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
JavaScript syntax:	-	*myRegExp* = RegExp
	-	*myRegExp* = new RegExp()
	-	*myRegExp* = RegExp(*aPattern*)
	-	*myRegExp* = RegExp(*aPattern*, *someAttribs*)
Argument list:	*aPattern*	A regular expression pattern
	someAttribs	One or more regular expression attributes
Class properties:	$n, index, input, lastMatch, lastParen, leftContext, multiline, rightContext	
Object properties:	$&, $', $*, $+, $_, $`, constructor, global, ignoreCase, index, lastIndex, prototype, source	
Object methods:	compile(), exec(), test(), toSource(), toString()	

The `RegExp` object implements some class (or static) methods which is fairly untypical of classes that support a constructor. There are also instance methods and properties.

Warnings:

- The static properties of a regular expression object do not conform to the same static scoping rules as the rest of JavaScript. Their static or class based properties are dynamically scoped and available in the scope chain from which they are executed. This is not the same as the scope rules for functions, which dictates that they run in the scope in which they are declared and not the scope from which they are called. This means that if a regular expression object is accessed in a function declared in one frame, when that function is called, the static properties are modified for the global built-in regular expression object that belongs to the calling frame. This avoids all manner of multithreaded simultaneous execution problems that would be difficult to deal with if the scoping rules for regular expression objects were the same as the rest of JavaScript.
- In Netscape Navigator, many properties of the `RegExp` built-in object are enumerable but they are not available in this way in MSIE.
- IE 5 does not properly support the `RegExp` object on the Macintosh platform. Many properties such as `lastMatch`, `leftContext`, etc. return an undefined value regardless of the `RegExp` result.

Example code:

```
// Create a RegExp object using a constructor
var myRegExp = new RegExp("sque[ea]ky", "g");
// Create the same RegExp object with a RegExp literal
var myRegExp = /sque[ea]ky/g;
```

See also:	Function scope, `RegExp` pattern –character literal, `RegExp()`, `RegExp.multiline`, `unwatch()`, `watch()`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`$&`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	Warning
`$'`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	Warning
`$*`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	Warning
`$+`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	Warning
`$_`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	-	-
`$``	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	Warning
`constructor`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	-	3 +	-
`global`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	ReadOnly
`ignoreCase`	1.2 +	5.5 +	4.0 +	5.5 +	5.0 +	3.0 +	3 +	ReadOnly
`index`	-	3.0 +	-	4.0 +	5.0 +	-	-	-
`lastIndex`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	Warning
`prototype`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	-	3 +	-
`source`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`compile()`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	-	-
`exec()`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	Warning
`test()`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	3.0 +	3 +	-
`toSource()`	1.3 +	-	4.06 +	-	-	-	-	-
`toString()`	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	-	3 +	-

Cross-references:

ECMA 262 edition 3 –section –15.1.4.8

ECMA 262 edition 3 –section –15.10.3

ECMA 262 edition 3 –section –15.10.4

RegExp() (Constructor)

A constructor function for creating new regular expression objects.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera 5.0	
JavaScript syntax:	-	`new RegExp(aPattern)`
	-	`new RegExp(aPattern, someAttribs)`
Argument list:	*aPattern*	A regular expression-matching pattern contained in a string
	someAttribs	A string containing the regular expression attributes

A `RegExp` constructor is used to create a new regular expression object containing a search pattern.

The first argument is a string containing a properly escaped pattern. Because the backslash character (\) is an escape when used in a string, for the back slashes to remain in the regular expression pattern, they will need to be double escaped. For example, this pattern:

```
/\d+/
```

Must be escaped like this when used in a string:

```
"\\d+"
```

The second argument is the attributes for the pattern. You should not put the attributes on the end of the pattern but specify them separately. These are available:

Operator	Description	JavaScript
`i`	Ignore case	JavaScript 1.2
`g`	Match globally	JavaScript 1.2
`ig`	Ignore case and match globally	JavaScript 1.2
`m`	Multiple line parsing	JavaScript 1.5

The regular expression constructor is useful for those occasions when you cannot easily predict what string you will need to match. If you could, it is likely you would use a regular expression literal. If the user is going to enter some value that determines what the search characteristics are to be then a `RegExp()` constructor may be useful.

Example code:

```
// Declare a variable and assign a RegExp literal to it
var myPattern = /x$/;
// Now do the same thing with a RegExp constructor
var myPattern = new RegExp("x$");
```

See also:	new, RegExp literal, `RegExp` object, RegExp pattern –attributes, RegExp pattern –character literal, `RegExp.constructor`, Regular expression

Cross-references:

ECMA 262 edition 3 –section –15.10.4

RegExp() (Function)

Another way to call the `exec()` method for the regular expression object.

Availability:	ECMAScript edition –3 JavaScript –1.2 Netscape –4.0	
JavaScript syntax:	N	`RegExp()`

See also:	`RegExp.$n`, `RegExp.input`, `RegExp.lastMatch`, `RegExp.lastParen`, `RegExp.leftContext`, `RegExp.multiline`, `RegExp.rightContext`, `RegExp.exec()`

Cross-references:

ECMA 262 edition 3 –section –15.10.3.1

RegExp.$_ (Property)

An alias for the input buffer of the regular expression.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myRegExp.$_`

Refer to:

`RegExp.input`

RegExp.$n (Property/static)

A property of the global `RegExp` object.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
JavaScript syntax:	-	`RegExp.$1`
	-	`RegExp.$2`
	-	`RegExp.$3`
	-	`RegExp.$4`
	-	`RegExp.$5`
	-	`RegExp.$6`
	-	`RegExp.$7`
	-	`RegExp.$8`
	-	`RegExp.$9`

There are nine similarly named properties whose name is a dollar symbol followed by a single digit, each of these holding text matched by a sub-expression (in parentheses) for the most recent match.

The ECMAScript edition 3 specification suggests that the range of values for this is $01 to $99 and for an implementation to be ECMA compliant, it should support that range of possibilities, as well as the $1 to $9 values.

Warnings:

- Since this is a class property (a static property), it belongs to the global built-in `RegExp` object. This means it is shared by all `RegExp` object instances and therefore is very transient and will be overwritten as soon as the next regular expression is evaluated. If you want to preserve the value, you will need to copy it immediately your regular expression has evaluated and before you call another.
- Note that there are only 9 of these. If you create a complex pattern that has more than 9 sub-expressions, you won't be able to access the sub-expressions above the ninth one unless the implementation is fully compliant with ECMAScript edition 3.
- Early versions of MSIE did not fully support this numbered property mechanism.

See also:	`RegExp` pattern –grouping, `RegExp` pattern –references, `RegExp.exec()`, `RegExp.test()`

Property attributes:

`ReadOnly.`

Cross-references:

ECMA 262 edition 3 –section –15.5.4.11

RegExp.compile() (Method)

Recompile a regular expression object's search mechanics.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
JavaScript syntax:	-	*myRegExp*.compile(*aPattern*)
	-	*myRegExp*.compile(*aPattern*, *someAttribs*)
Argument list:	*aPattern*	A regular expression-matching pattern contained in a string
	someAttribs	A string containing the regular expression attributes

This takes the same argument as the constructor and can be used to recycle an existing `RegExp` object with a freshly initialized search pattern and attributes.

The first argument is a string containing a properly escaped pattern. Because the backslash character (\) is an escape when used in a string, for the backslashes to remain in the regular expression pattern, they will need to be double escaped. For example, this pattern:

```
/\d+/
```

Must be escaped like this when used in a string:

```
"\\d+"
```

The second argument is the attributes for the pattern. You should not put the attributes on the end of the pattern but specify them separately. These are available:

Operator	Description	JavaScript
i	Ignore case	JavaScript 1.2
g	Match globally	JavaScript 1.2
ig	Ignore case and match globally	JavaScript 1.2
m	Multiple line parsing	JavaScript 1.5

The regular expression `compile()` method is useful for those occasions when you cannot easily predict what string you will need to match. As is the case with the constructor you can define the pattern based on a user-specified value. With the `compile()` method, you can replace the pattern as needed without destroying and recreating the object again.

Calling the `compile()` method on a `RegExp` object is a good way to gain some performance improvements if the regular expression is used frequently during a session. It will need compiling again each time it is instantiated.

RegExp.constructor (Property)

A reference to the constructor object for regular expressions.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera 5.0	
Property/method value type:	`RegExp` object	
JavaScript syntax:	-	*myRegExp*`.constructor`

The constructor is that of the built-in `RegExp` prototype object.

You can use this as one way of creating regular expressions although it is more popular to use the new `RegExp()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape Navigator provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	`RegExp()`

Cross-references:

ECMA 262 edition 3 –section –15.10.6.1

RegExp.exec() (Method)

This performs a non-destructive match on a target string.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
Property/method value type:	`Array` object	
JavaScript syntax:	-	*myRegExp*`.exec()`
	-	*myRegExp*`.exec(`*aString*`)`
Argument list:	*aString*	A string object to run a regular expression match against

This is functionally similar to the `String.match()` method. It returns an enhanced array object in the same way. You would use this if you need to know the location of the matched string and whether it occurs more than once.

The `g` attribute is ignored for `RegExp.exec()` methods unlike the `String.match()` method.

When the `exec()` method is received by a global `RegExp` object, the `lastIndex` property of that `RegExp` object is set to point at a character location immediately following the previous match. This means you can use sub-stringing techniques to walk through the string calling `exec()` in an iterator until a `null` value is returned. This lets you build an iterator based on the pattern and the number of times it recurs in the searched string. The `RegExp.exec()` method does most of the work for you since it continues where it left off during the previous search. This also means you can reset the search point or commence searching wherever you like in the target string.

A shortcut mechanism to calling the `exec()` method is to call the `RegExp` itself as a function. Thus we can create a regular expression object and a target string object:

```
myRegExp = new RegExp("/\\d+/");

myString = "aaa 111 bbb 222 ccc";
```

We can execute the regular expression either like this:

```
myRegExp.exec(myString);
```

Or like this:

```
myRegExp(myString);
```

The result of calling this method is the `null` value if no match occurs. Otherwise, an array as per the `String.match()` method is returned.

The array object has an additional property named `index`, which contains the character location where the match occurred. It also has an additional property called `input`, which contains the original string that was searched for a match.

Warnings:

- Support for this method is bugged in IE 4.
- If you do not pass a string to the `RegExp.exec()` method, it will match against the current value of the `RegExp.input` property that belongs as a static class property of the built-in `RegExp` object.
- The input property gets set automatically by client-side event handlers for `FormElement` objects in a web page form.

See also:	`Array.index`, `Array.input`, `RegExp` pattern, `RegExp.$n`, `RegExp.input`, `RegExp.lastIndex`, `RegExp.lastMatch`, `RegExp.lastParen`, `RegExp.leftContext`, `RegExp.multiline`, `RegExp.rightContext`, `RegExp.test()`, Regular expression, `String.match()`, `String.replace()`, `String.search()`, `String.split()`

Cross-references:

ECMA 262 edition 3 –section –15.10.6.2

RegExp.global (Property)

An instance property of a regular expression object.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myRegExp*`.global`

A Boolean value that indicates the state of the `g` attribute on the regular expression pattern.

Property attributes:

`ReadOnly`.

Cross-references:

ECMA 262 edition 3 –section –15.10.7.2

RegExp.ignoreCase (Property)

An instance property of a regular expression object.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3 Internet Explorer –4 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myRegExp*.ignoreCase

A Boolean value that indicates the state of the `i` attribute on the regular expression pattern.

Property attributes:

`ReadOnly`.

Cross-references:

ECMA 262 edition 3 –section –15.10.7.3

RegExp.index (Property)

The position of the first match in the string.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myRegExp*.index

A character index position into the source string where the first match is located.

RegExp.input (Property/static)

A property of the global `RegExp` object.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera 5.0	
JavaScript syntax:	-	*RegExp*.input

A default string to search when the `exec()` and `test()` methods are called with no arguments.

You can set this and then call the `RegExp.exec()` or `RegExp.test()` methods without passing a string argument value. They will then use this property of the built-in global `RegExp` object as their target search string.

Warnings:

- This behaves slightly differently in JavaScript version 1.2 in Netscape Navigator.
- Since this is a class property (a static property), it belongs to the global built-in `RegExp` object. This means it is shared by all `RegExp` object instances and therefore is very transient and will be overwritten as soon as the next regular expression is evaluated. If you want to preserve the value, you will need to copy it immediately your regular expression has evaluated and before you call another.
- In Netscape 4, event handlers for `FormElement` objects in a web page form automatically load the input property of the built-in `RegExp` object as the event handlers are fired. Hence, you should not rely on the input property remaining consistent once your function call exits. Set it immediately before you need to parse the string and do not bank on it being there later.
- This property was not supported fully in IE 4 as it was read-only. The work-around was to always pass in a value to the `exec()` method when it is invoked.

See also:	`RegExp.exec(),RegExp.test()`

RegExp.lastIndex (Property)

A character index within the searched string immediately following the previous match.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`myRegExp.lastIndex`

The `lastIndex` property is set to point at the character following the previous match. This allows you to build an iterator that cycles according to the matches for the search pattern within the source string.

When a `null` result is returned from the `RegExp.exec()` method, this value is set to 0.

The `RegExp.exec()` method uses the `lastIndex` property as its starting point for the next search. This means that if you prematurely exit the pattern searching iterator, you should also manually set this value to 0 as well. This will allow the same `RegExp` object to be used with another pattern (or the same one) to search a different string.

Warnings:

- The `lastIndex` property will be reset to 0 when the regular expression is called by `String.search()`,`String.replace()` and `String.match()`.
- The `lastIndex` behaviour is only appropriate when the `g` attribute is applied to the pattern that searches the target string. It is of no consequence in a non-global pattern match.

See also:	`RegExp.exec()`,`String.match()`,`String.replace()`, `String.search()`

Cross-references:

ECMA 262 edition 3 –section –15.10.7.5

RegExp.lastMatch (Property/static)

A property of the global `RegExp` object.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
JavaScript syntax:	-	`RegExp.lastMatch`

This property returns the most recently matched text.

Warnings:

- Since this is a class property (a static property), it belongs to the global built-in `RegExp` object. This means it is shared by all `RegExp` object instances and therefore is very transient and will be overwritten as soon as the next regular expression is evaluated. If you want to preserve the value, you will need to copy it immediately after your regular expression has been evaluated and before you call another.

See also:	`RegExp.exec()`,`RegExp.test()`,`RegExp["$&"]`

Property attributes:

`ReadOnly`.

RegExp.lastParen (Property/static)

A property of the global `RegExp` object.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
JavaScript syntax:	-	`RegExp.lastParen`

This property returns the text matched by the last sub-expression which was part of the most recent match. It will contain a `null` or undefined value if there was no previous match.

Warnings:

- Since this is a class property (a static property), it belongs to the global built-in `RegExp` object. This means it is shared by all `RegExp` object instances and therefore is very transient and will be overwritten as soon as the next regular expression is evaluated. If you want to preserve the value, you will need to copy it immediately after your regular expression has been evaluated and before you call another.

See also:	`RegExp` pattern –grouping, `RegExp` pattern –references, `RegExp.exec()`,`RegExp.test()`,`RegExp["$+"]`

Property attributes:

`ReadOnly`.

RegExp.leftContext (Property/static)

A property of the global `RegExp` object.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
JavaScript syntax:	-	`RegExp.leftContext`

This property returns the text to the left of the most recent match.

Warnings:

- Since this is a class property (a static property), it belongs to the global built-in `RegExp` object. This means it is shared by all `RegExp` object instances and therefore is very transient and will be overwritten as soon as the next regular expression is evaluated. If you want to preserve the value, you will need to copy it immediately after your regular expression has been evaluated and before you call another.

See also:	`RegExp.exec()`,`RegExp.test()`,`RegExp["$`"]`

Property attributes:

`ReadOnly`.

RegExp.multiline (Property/static)

A regular expression attribute to control the scope of the pattern.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`RegExp`*`.multiline = `*`aBoolean`*
Argument list:	*`aBoolean`*	A switch to set the property `true` or `false`

When you want the regular expression to apply to multiple lines separated by newline characters, this property should be set to `true`. When it is `false` the pattern is applied only to a single line.

Warnings:

- In JavaScript version 1.2 in Netscape Navigator, this is the only way to set the pattern to operate on multiple lines.
- The 'm' attribute is provided in JavaScript version 1.3 in some implementations.
- Note that client-side JavaScript used in Netscape 4 will set the multi-line property to `true` when it is used in an event handler for a `TextArea` object. The value is restored to whatever it was before exiting the handler. This suggests that you should explicitly set the multi-line property to the value you require when to parse a regular expression. You cannot rely on the stability of any value you may have stored in that attribute previously.
- Since this is a class property (a static property), it belongs to the global built-in `RegExp` object. This means it is shared by all `RegExp` object instances and therefore is very transient and will be overwritten as soon as the next regular expression is evaluated. If you want to preserve the value, you will need to copy it immediately before your regular expression has been evaluated and before you call another.

Example code:

```
RegExp.multiline = true;
RegExp.multiline = false;
```

See also:	`RegExp` object, `RegExp` pattern –attributes, `RegExp` pattern – position, `RegExp.exec()`, `RegExp.test()`, `RegExp["$*"]`

Cross-references:

ECMA 262 edition 3 –section –15.10.7.4

RegExp.prototype (Property)

The prototype for the `RegExp` object that can be used to extend the interface for all `RegExp` objects.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera –5.0	
Property/method value type:	`RegExp object`	
JavaScript syntax:	-	`RegExp.prototype`
	-	*`myRegExp`*`.constructor.prototype`

You can use this to extend the interface to the object class and provide additional capabilities to your regular expressions.

See also:	`prototype` property, `RegExp.toString()`

Cross-references:

ECMA 262 edition 3 –section –15.10.5.1

ECMA 262 edition 3 –section –15.10.6.1

RegExp.rightContext (Property/static)

A property of the global `RegExp` object.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5	
JavaScript syntax:	-	`RegExp.rightContext`

This property returns the text to the right of the most recent match.

Warnings:

- Since this is a class property (a static property), it belongs to the global built-in `RegExp` object. This means it is shared by all `RegExp` object instances and therefore is very transient and will be overwritten as soon as the next regular expression is evaluated. If you want to preserve the value, you will need to copy it immediately after your regular expression has been evaluated and before you call another.

See also:	`RegExp.exec(),RegExp.test(),RegExp["$'"]`

Property attributes:

`ReadOnly`.

RegExp.source (Property)

An instance property of a regular expression object.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myRegExp*`.source`

This is a read-only property that returns the regular expression pattern text.

Property attributes:

`ReadOnly`.

Cross-references:

ECMA 262 edition 3 –section –15.10.7.1

RegExp.test() (Method)

Another name for the `RegExp.exec()` method when used with global `RegExp` objects.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myRegExp*`.test(`*aString*`)`
Argument list:	*aString*	The string to be examined by the regular expression

This is not quite like the `exec()` method. In this case, you cannot establish whether the match occurs more than once or where in the string the match occurs.

This method simply returns `true` or `false` to indicate whether the pattern matches anything in the search string.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
document.write("1 "     + testForYearNumber("1")     + "<BR>");
document.write("12 "    + testForYearNumber("12")    + "<BR>");
document.write("123 "   + testForYearNumber("123")   + "<BR>");
document.write("1234 "  + testForYearNumber("1234")  + "<BR>");
document.write("12345 " + testForYearNumber("12345") + "<BR>");
function testForYearNumber(aString)
{
   var myRegExp = /^\d{4}$/;
   return myRegExp.test(aString);
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`RegExp.$n`, `RegExp.exec()`, `RegExp.input`, `RegExp.lastMatch`, `RegExp.lastParen`, `RegExp.leftContext`, `RegExp.multiline`, `RegExp.rightContext`

Cross-references:

ECMA 262 edition 3 –section –15.10.6.3

RegExp.toSource() (Method)

Outputs a regular expression object formatted as a `RegExp` literal contained in a string.

Availability:	JavaScript –1.3 Netscape –4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*`myRegExp`*`.toSource()`

This is an alternative way to deliver a string version of a regular expression object. In this case, it is formatted as a `RegExp` literal and can then be used in an `eval()` function to assign another regular expression.

RegExp.toString() (Method)

Returns a string primitive version of a `RegExp` object.

Availability:	ECMAScript edition –3 JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0 Opera –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myRegExp`*`.toString()`

This method returns a String primitive representation of the value of the present `RegExp` value.

See also:	Cast operator, `RegExp.prototype.toString()`

Cross-references:

ECMA 262 edition 3 –section –15.10.6.4

RegExp["$&"] (Property)

An alternative way to refer to the text of the most recent successful pattern match.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myRegExp*`["$&"]`

This property uses the array index notation because the property name contains punctuation characters that would cause syntactic errors if the property name were not in quotes.

Warnings:

- This is provided as a convenience to Perl programmers who may be familiar with its syntax in this form. However, it is recommended that you use the named property form when assigning values to regular expression objects.

See also:	`RegExp.lastMatch`

Property attributes:

`ReadOnly`.

RegExp["$'"] (Property/static)

An alternative means of referring to the text to the right of the most recent match.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5.0	
JavaScript syntax:	-	`RegExp["$'"]`

This property uses the array index notation because the property name contains punctuation characters that would cause syntactic errors if the property name were not in quotes.

Warnings:

- This is provided as a convenience to Perl programmers who may be familiar with its syntax in this form. However, it is recommended that you use the named property form when assigning values to regular expression objects.

See also:	`RegExp.rightContext`

Property attributes:

`ReadOnly`.

RegExp["$*"] (Property/static)

A switch property to determine whether multi-line matching is performed or not.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5	
JavaScript syntax:	-	`RegExp["$*"]`

This property uses the array index notation because the property name contains punctuation characters that would cause syntactic errors if the property name were not in quotes.

Warnings:

- This is provided as a convenience to Perl programmers who may be familiar with its syntax in this form. However, it is recommended that you use the named property form when assigning values to regular expression objects.

See also:	`RegExp.multiline`

RegExp["$+"] (Property/static)

An alias for the text that matches the most recent sub-expressions.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5	
JavaScript syntax:	-	`RegExp["$+"]`

This property uses the array index notation because the property name contains punctuation characters that would cause syntactic errors if the property name were not in quotes.

Warnings:

- This is provided as a convenience to Perl programmers who may be familiar with its syntax in this form. However, it is recommended that you use the named property form when assigning values to regular expression objects.

See also:	`RegExp.lastParen`

Property attributes:

`ReadOnly`.

RegExp["$`"] (Property/static)

An alias for the text to the left of the most recent match.

Availability:	JavaScript –1.2 JScript –5.5 Internet Explorer –5.5 Netscape –4.0 Netscape Enterprise Server –3.0 Opera –5	
JavaScript syntax:	-	`RegExp["$`"]`

This property uses the array index notation because the property name contains punctuation characters that would cause syntactic errors if the property name were not in quotes.

Warnings:

- This is provided as a convenience to Perl programmers who may be familiar with its syntax in this form. However, it is recommended that you use the named property form when assigning values to regular expression objects.

See also:	`RegExp.leftContext`

Property attributes:

`ReadOnly`.

RegExp pattern (Definition)

The sequence of characters that comprise a regular expression.

Availability:	ECMAScript edition –3

The rules for creating patterns are complex and involved, hence this topic has been broken into several related sub-topics.

The syntax is similar and almost as complete as that used for Perl.

A pattern is constructed using a sequence of characters. Most of the typeable characters can be used as matches for that particular character, but some special characters are used in the regular expression to mean something more abstract. Some characters need to be escaped, however they are not escaped in the same way as the normal escape sequences that are used in string literals. The literal characters are described in their own sub-topic.

There are other special character sequences, called meta-characters, that describe groups or classes of characters; these are also described in their own topic. They are called meta-characters.

You can build repetitions of literal characters and special character classes, which is also covered in a sub-topic.

See also:	`RegExp` pattern –alternation, `RegExp` pattern –attributes, `RegExp` pattern –character class, `RegExp` pattern –character literal, `RegExp` pattern –grouping, `RegExp` pattern –position, `RegExp` pattern –references, `RegExp` pattern –repetition, `RegExp` pattern –sub-patterns, `RegExp.exec()`, Regular expression, `String.match()`,`String.replace()`,`String.search()`,`String.split()`

Cross-references:

ECMA 262 edition 3 –section –15.10.1

RegExp pattern –alternation (Definition)

Sometimes you will want to match either one pattern or another using alternatives.

Availability:	ECMAScript edition –3

You can group sub-expressions and offer them as alternatives. For example, you might want to match "AB" or "CD". The alternation operator can be placed between each sub-expression to indicate choices like this:

```
/AB|CD/
```

This technique can be used in combinations with character literals, classes and repetition operators. Like this:

```
/[a-z]{3}|[0-9]{40}/
```

This matches with three lower case letters or 40 numeric digits and nothing else will match.

See also:	`RegExp` pattern, `RegExp` pattern –character class, `RegExp` pattern –character literal, `RegExp` pattern –repetition

Cross-references:

ECMA 262 edition 3 –section –15.10.1

ECMA 262 edition 3 –section –15.10.2.3

ECMA 262 edition 3 –section –15.10.2.4

RegExp pattern –attributes (Definition)

The attributes of a regular expression.

Availability:	ECMAScript edition –3

When you specify a regular expression, it will by default only match the first occurrence that it encounters. It is possible to specify a pattern to match that can occur several times in a line. To match all of these, you can use the '`g`' attribute. The '`g`' stands for global matching.

There is one other attribute that allows the matching to be carried out in a case-insensitive manner. That is the '`i`' attribute.

Both of these attributes are placed after the closing slash at the end of the pattern.

The case-insensitive match works like this:

```
/javascript/i
```

would match the following strings

```
JavaScript
```

```
javascript
```

```
JAVASCRIPT
```

```
JaVaScRiPt
```

The `g` attribute applied like this, would cause the pattern:

```
/0/g
```

to match every zero in the string:

```
'0100, 00123, "00067", 666000'
```

There is another attribute to control whether the pattern is applied to single lines or multiple lines. Because the regular expression is realized as an object, this is controlled by means of an object property accessed via the multi-line identifier, the letter 'm' in JavaScript 1.5 (shipped with Netscape 6.0)

Here is a list of the available attributes:

seq	Pattern	Description	JS
00	g	Global match for every occurrence of the pattern throughout the line.	1.2
01	i	Case-insensitive matching.	1.2
02	ig	Case-insensitive global match.	1.2
13	m	Multiple lines to be processed. Available only in JavaScript 1.5 and mutually exclusive with the 's' attribute.	1.5
15	gm	Global match on multiple lines.	1.5
15	im	Case-insensitive matching on multiple lines.	1.5
16	igm	Case-insensitive global match on multiple lines.	1.5

See also:	RegExp pattern, RegExp pattern –position, RegExp(), RegExp.multiline

Cross-references:

ECMA 262 edition 3 –section –15.10.1

ECMA 262 edition 3 –section –15.10.2.1

RegExp pattern –character class (Definition)

RegExp pattern components for describing character classes.

Availability:	ECMAScript edition –3

Character classes are a group of RegExp character literals enclosed in square brackets. This set of characters can then be placed into a RegExp pattern to provide a match for any character that is considered to be a member of the class.

You can use character classes to include or exclude groups of characters. An exclusion is called a negated class and is signified by placing a circumflex (^) character as the first one inside the square brackets. Thus, [abcd] will match true if the character being tested at that position in the pattern is the letter "a", "b", "c" or "d".

The character class [^abcd] will match any character as long as it is not one of those four.

You can indicate a range of characters by using a hyphen. Therefore [abcd] is the same as saying [a-d]. Other examples are summarized in the following table.

There are special character classes already built in so that you don't have to create your own. These are done using escaped characters very similar to those in the literal character set. These can also be used inside the square brackets to construct other class groups of characters.

Pattern	Description
`[ ... ]`	Any single character that is one of the set enclosed in the square brackets.
`[^ ... ]`	Any single character that is not one of the set enclosed in the square brackets.
`[^abcd]`	Any character that is not one of the letters "a", "b", "c" or "d".
`[abcd]`	Any one of the letters "a", "b", "c" or "d".
`[a-z]`	Any single lower case character.
`[A-Z]`	Any single upper case character.
`[a-zA-Z]`	Any single alphabetic character.
`[0-7]`	Any octal numeric digit.
`.`	Any character apart from newline.
`\d`	Any decimal digit character.
`\s`	Any whitespace character.
`\w`	Any word character (which is any letter, number or underscore). This does not mean a whitespace character.
`\D`	Any non-digit character.
`\S`	Any non-whitespace character. This is not necessarily a valid word character.
`\W`	Any non-word character.
`[\b]`	A literal backspace not to be confused with a word boundary match (using the \b outside of the square brackets)
`[0-1]`	Any binary numeric digit.
`[0-9A-F]`	Any hexadecimal numeric digit.
`[\dA-F]`	Any hexadecimal numeric digit (alternative form).
`[a-zA-Z0-9]`	Any single alphanumeric character.
`[a-zA-Z\d]`	Any single alphanumeric character (alternative form).
`[^a-zA-Z0-9_\$]`	Any character that is not valid within an identifier name.
`[a-zA-Z0-9_\$]`	Any character that is valid within an identifier name.
`[0-9]`	Any decimal numeric digit, (alternative version).
`[^0-9]`	Any any character that is not a digit, (alternative version).
`[\t\n\r\f\v]`	Any whitespace character, (alternative version).
`[^\t\n\r\f\v]`	Any non-whitespace character, (alternative version).
`[^\n]`	Any character apart from newline, (alternative version).
`[^a-zA-Z0-9_]`	Any non-word character, (alternative version).
`[a-zA-Z0-9_]`	Any word character, (alternative version).

Warnings:

- Beware of the backspace escape `\b`. To match against a backspace it must be enclosed in square brackets to create a character class. If it is used outside of the brackets, it is interpreted to mean a word boundary.

See also:	`RegExp` pattern, `RegExp` pattern –alternation, `RegExp` pattern – character literal

Cross-references:

ECMA 262 edition 3 –section –15.10.1

ECMA 262 edition 3 –section –15.10.2.6

ECMA 262 edition 3 –section –15.10.2.12

RegExp pattern –character literal (Definition)

These are the characters that can be used in the pattern to match against themselves.

Availability:	ECMAScript edition –3

The literal characters in a regular expression denote a particular character and have no special meaning other than to match that character at the position in the string they are defined within the overall pattern.

Simple, fixed and constant patterns can be defined purely with literal characters.

The string "JavaScript" can be matched exactly and exclusively with the pattern `/JavaScript/`. Unless some additional information is added to the pattern description, neither `/javascript/` nor `/JAVASCRIPT/` will match. However, the pattern `/Java/` will match strings containing both "JavaScript" and "Java".

Here is a list of the literal characters as they would be used in the pattern:

Pattern	Description
`0` to `9`	Itself
`a` to `z`	Itself
`A` to `Z`	Itself
`\$`	A single dollar sign ($)
`\*`	A single asterisk (*)
`\+`	A single plus sign (+)
`\,`	A single comma (,)
`\.`	A single period (.)

Table continued on following page

Pattern	Description
`\/`	A single slash (/)
`\?`	A single question mark (?)
`\\`	A single backslash (\)
`\^`	A single circumflex (^)
`\d`	Any digit character as per [0-9]
`\D`	Any non-digit character as per [^0-9]
`\f`	A form feed
`\n`	A newline
`\r`	A carriage return
`\S`	A non-space character
`\s`	A space character
`\t`	A tab character
`\v`	A vertical tab
`\w`	An alphanumeric character and underscore as per [0-9a-zA-Z_]
`\W`	An non-alphanumeric character and underscore as per [^0-9a-zA-Z_]
`\\|`	A single vertical bar (|)
`\(`	A single opening parenthesis (()
`\)`	A single closing parenthesis ())
`\[`	A single opening square bracket ([)
`\]`	A single closing square bracket (])
`\{`	A single opening curly brace ({)
`\}`	A single closing curly brace (})
`\nnn`	The ASCII character encoded by the octal value nnn
`\onnn`	The ASCII character encoded by the octal value nnn
`\uhhhh`	The Unicode character encoded by the hexadecimal value hhhh
`\xhh`	The ASCII character encoded by the hexadecimal value hh
`\c*`	The control character equivalent to ^*
`\c@`	(NUL) –Null character
`\c[`	(ESC) –Escape
`\c\`	(FS) –File separator (Form separator)
`\c]`	(GS) –Group separator
`\c^`	(RS) –Record separator
`\c_`	(US) –Unit separator
`\cA`	(SOH) –Start of header
`\cB`	(STX) –Start of text
`\cC`	(ETX) –End of text
`\cD`	(EOT) –End of transmission
`\cE`	(ENQ) –Enquiry

Table continued on following page

Pattern	Description
\cF	(ACK) –Positive acknowledge
\cG	(BEL) –Alert (bell)
\cH	(BS) –Backspace
\cI	(HT) –Horizontal tab
\cJ	(LF) –Line feed
\cK	(VT) –Vertical tab
\cL	(FF) –Form feed
\cM	(CR) –Carriage return
\cN	(SO) –Shift out
\cO	(SI) –Shift in
\cP	(DLE) –Data link escape
\cQ	(DC1) –Device control 1 (XON)
\cR	(DC2) –Device control 2 (tape on)
\cS	(DC3) –Device control 3 (XOFF)
\cT	(DC4) –Device control 4 (tape off)
\cU	(NAK) –Negative acknowledgement
\cV	(SYN) –Synchronous idle
\cW	(ETB) –End of transmission block
\cX	(CAN) –Cancel
\cY	(EM) –End of medium
\cZ	(SUB) –Substitute
\0 to \9	The last remembered substring as per the $n property
[\b]	A literal backspace not to be confused with a word boundary match (using the \b outside of square brackets)

It is necessary to escape the punctuation characters as they assume special meanings when used in a pattern on their own. Refer to the other RegExp topics for further details.

See also:	RegExp literal, RegExp object, RegExp pattern, RegExp pattern –alternation, RegExp pattern –character class, RegExp(), Regular expression

Cross-references:

ECMA 262 edition 3 –section –15.10.1

ECMA 262 edition 3 –section –15.10.2.2

ECMA 262 edition 3 –section –15.10.2.10

ECMA 262 edition 3 –section –15.10.2.11

RegExp pattern –extension syntax (Definition)

A Perl 5 capability for specifying anchors with look-ahead patterns.

Availability:	ECMAScript edition –3

Positional anchors can be constructed using patterns. These are enclosed in grouping operators and commence with a question mark.

A logical negation is also possible. However the syntax for this uses the JavaScript convention of an exclamation mark (`!`) rather than the regular expression convention of a circumflex (`^`).

A positional anchor, marking the location immediately before a capital letter would look like this:

```
(?=[A-Z])
```

A positional anchor marking a location immediately before any non-capital letter would look like this:

```
(?![A-Z])
```

The second example uses the logical negation operator.

Note that these are positional anchors that behave like the `^` and `$` which denote the start and end of a line. They do not select any characters.

See also:	`RegExp` pattern –grouping, `RegExp` pattern –position, `RegExp` pattern –references

Cross-references:

ECMA 262 edition 3 –section –15.10.1

ECMA 262 edition 3 –section –15.10.2.7

ECMA 262 edition 3 –section –15.10.2.14

RegExp pattern –grouping (Definition)

Sometimes you may want to group several items to treat them conditionally or repetitively. The grouping operator provides the means to do that.

Availability:	ECMAScript edition –3

The parentheses grouping operator works inside regular expressions much as you would expect it to. Everything inside the parentheses can be operated on by a repetition. This example matches the word java or javascript:

```
/java(script)?/
```

The grouping operator is also used to delimit sub-strings.

See also:	`RegExp` pattern, `RegExp` pattern –extension syntax, `RegExp` pattern –references, `RegExp` pattern –sub-patterns, `RegExp.$n`, `RegExp.lastParen`

Cross-references:

ECMA 262 edition 3 –section –15.10.1

ECMA 262 edition 3 –section –15.10.2.8

RegExp pattern –position (Definition)

Aligning the pattern to one or other end of the string sometimes helps to remove ambiguity from the match.

Availability:	ECMAScript edition –3

Occasionally, you may want to match a string that is explicitly at the front or back end of a line. You can do this by adding the special characters to the expression to mark the location of the start of the line or the end of the line. You can also align the pattern with word boundaries as well. Here is a summary of some special patterns that are position aligned:

Pattern	Description
`^`	Indicate the start of the line
`$`	Indicate the end of the line
`\b`	Indicate a word boundary. Note that this cannot be used in a bracketed character class [\b] means backspace not word boundary
`\B`	Indicate any non-word boundary location
`.$`	The last character at the end of the line (the dot matches one character)
`\b\d*\b`	A complete word composed only of numeric digits
`\b\w*\b`	A complete word
`\s*$`	All of the trailing whitespace
`^$`	A line with nothing between the start and end, an empty line
`^.`	The first character at the beginning of the line (the dot matches one character)
`^.*$`	The entire line regardless of its contents
`^\s`	A leading whitespace character

Warnings:

- In multi-line mode, the markers for the start and end of the line apply to each line in turn.

See also:	`RegExp` pattern, `RegExp` pattern –attributes, `RegExp` pattern –extension syntax, `RegExp.multiline`

Cross-references:

ECMA 262 edition 3 –section –15.10.1

RegExp pattern –references (Definition)

Groups of characters in a pattern can be referred to symbolically later in the expression.

Availability:	ECMAScript edition –3

We can use parentheses to repeat a match so that the same text is matched in more than one place in the expression. It is not the pattern that is repeated but the matched value. We do this by referring to the sub-expression according to its indexed location within the pattern. The index is incremented for every left parenthesis encountered. The first is referred to as `\1`, the second as `\2` and so on. This is useful for balancing matching quote symbols around a text string that might have either single or double quote marks around it. Unless we can relate the matches at each end, we might find that we do in fact have either one of the two quote symbols but we don't necessarily have a balanced pair.

This matches a single or double quote character:

```
/['"]/
```

This matches a single or double quote character at either end of any other character sequence:

```
/['"].*['"]/
```

However, the text between the quotes could contain a quote so we'll replace the match between them with any non-quote character. Like this:

```
/['"][^'"]*['"]/
```

To use a reference to a previous sub-expression, we need to mark the sub-expression with parentheses:

```
/(['"])[^'"]*['"]/
```

Now we can refer to the first sub-expression at a later stage and require that the same characters be repeated:

```
/(['"])[^'"]*\1/
```

This ensures that the sequence of characters `'AAA'` would match but `'AAA"` would not even though `"AAA"` would.

A grouped sub-expression can be prevented from being indexed by placing a question mark and colon immediately inside the parentheses and then the item cannot be indexed as a reference. This works in JavaScript version 1.3 onwards.

Warnings:

- Beware that you do not use an index greater than the number of sub-patterns available. The backslash and digit will be interpreted as a character escape otherwise.
- Because the backslash can also be interpreted as a character escape you must be unambiguous as to your intentions. To ensure that it is interpreted as a character escape, use a value that is more than one digit long.

See also:	`RegExp` pattern, `RegExp` pattern –extension syntax, `RegExp` pattern –grouping, `RegExp` pattern –sub-patterns, `RegExp.$n`, `RegExp.lastParen`

Cross-references:

ECMA 262 edition 3 –section –15.10.1

RegExp pattern –repetition (Definition)

Parts of a matching pattern can be repeated for multiple characters.

Availability:	ECMAScript edition –3

We can create matching patterns that will match a fixed number of characters. For example, a pattern for a zero padded decimal number that is 4 digits long would be:

```
/\d\d\d\d/
```

Or it could be expressed like this:

```
/[0-9][0-9][0-9][0-9]/
```

Or even like this:

```
/[0123456789][0123456789][0123456789][0123456789]/
```

These are fine for fixed length formatted numbers but very often the formatting of some text can be variable and unless numbers are zero padded, we don't know how long they are. We can extend a matching pattern to signify that it should match a certain number of times or an unspecified and variable number of times.

The curly braces are used to indicate a repetition count. A single value indicates a repeat count that is a fixed length. So our first example could be expressed like this:

```
/\d{4}/
```

If the curly braces include a pair of comma separated values, the first is a minimum number and the second is a maximum. So if we wanted to match a value that was between 10 and 1000 and it was not zero padded, we could do this:

```
/\d{2,4}/
```

UK style postcodes are formatted in general with a standard layout. Mostly they conform to the pattern:

```
AANN NNAA
```

That is two letters, up to two numbers, a space and then up to two numbers and two letters.

You could match a UK style postcode with something like this:

```
/\w{2}\d{1,2}\s\d{1,2}\w{2}/
```

A United States Zip code is simpler, being made up of two letters, a space and 5 digits. So it could be matched with:

```
/\w\w\s\d{5}/
```

There is a way to indicate that characters are optional with the question mark character (`?`).

The UK postcode example could then be simplified to allow the second character group to be optional as could the inner space character. Like this:

```
/\w\w\d\d?\s?\d?\d?\w?\w?/
```

The plus sign is used to match one or more instances of the character to its left and the asterisk to match zero or more occurrences. Here are some examples:

Seq	Pattern	Description
01	`{a,b}`	Match the item to the left between `a` and `b` times.
02	`{a,}`	Match the item to the left at least `a` times or more.
03	`{a}`	Match the item to the left exactly `a` times, no more, no less.
04	`?`	Match the item to the left zero or one times.
05	`+`	Match the item to the left 1 or more times.
05	`+?`	Match the item to the left 1 or more times using a minimal matching technique. (JavaScript 1.3)
06	`*`	Match the item to the left zero or more times.
06	`*?`	Match the item to the left zero or more times using a minimal matching technique. (JavaScript 1.3)
99	`{0,1}`	Match the item to the left zero or one times (alternative form).
99	`{0,}`	Match the item to the left zero or more times (alternative form).
99	`{1,}`	Match the item to the left 1 or more times (alternative form).

The minimal matching technique is implemented in JavaScript version 1.3 and is based on the facilities of Perl version 5 interpreters. Minimal matching is where a match occurs with the minimum number of characters necessary to make a match. This is as opposed to the normal technique, which matches as many characters as possible.

See also:	`RegExp` pattern, `RegExp` pattern –alternation

Cross-references:

ECMA 262 edition 3 –section –15.10.2.5

ECMA 262 edition 3 –section –15.10.10

RegExp pattern –sub-patterns (Definition)

When a pattern matches, it is possible to extract a portion of that for re-use.

Availability:	ECMAScript edition –3

By using parentheses creatively in a regular expression a sub-pattern can be marked out such that when a match occurs, that sub-pattern can be extracted.

Here is a pattern that matches a date format:

```
/[0-9]{2}-[a-zA-Z]{3}-[0-9]{4}/
```

We might want to extract just the month name. We can place parentheses around that so it can be extracted later. Like this:

```
/[0-9]{2}-([a-zA-Z]{3})-[0-9]{4}/
```

The example shows how this works:

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Make the RegExp from a literal
myRegExp = /[0-9]{2}-([a-zA-Z]{3})-[0-9]{4}/;
// Create the input search string
myString = "01-Jan-1954";
// Run the search
myArray = myRegExp.exec(myString);
// How many items are there in the array
document.write(myArray.length);
document.write("<BR>");
// Display the source string
document.write(myArray[0]);
document.write("<BR>");
// Display the month name sub-match
document.write(myArray[1]);
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`RegExp` pattern, `RegExp` pattern –grouping, `RegExp` pattern –references

Cross-references:

ECMA 262 edition 3 –section –15.10.1

Regular expression (Definition)

A means of matching patterns of text in string values.

Regular expressions are a way of describing a pattern match that can be used to select a group of characters from an input string. This usually then leads on to replacing them with some other set of characters. It is analogous to the find and replace capability in a word processor.

JavaScript version 1.2 introduced regular expression support by way of a specialized object, a utility that UNIX developers have long known about and used. It has migrated into many desktop applications now and has become a somewhat portable way of matching text strings with one another and performing edits on them. The regular expression syntax adopted by JavaScript version 1.2 emulates that which is commonly used in Perl interpreters, specifically the syntax that is supported is generally called Perl version 4. In JavaScript version 1.3, the regular expression syntax was expanded to support Perl version 5 syntax.

Regular expressions are managed by creating a `RegExp` object with the `RegExp()` constructor. `RegExp` objects support a literal syntax and can be created on the fly without needing a constructor call which makes them extremely convenient to deploy.

The ECMAScript standard ratifies regular expressions in the third edition. They were not present in the second edition.

Warnings:

- Regular expressions are still an area of concern when developing portable content. For example, they are completely unsupported on the WebTV platform as of the Summer 2000 release of the JellyScript interpreter.

See also:	Fundamental data type, JellyScript, `RegExp` pattern, `RegExp` pattern –character literal, `RegExp()`, `RegExp.exec()`, `String.match()`, `String.replace()`, `String.search()`, `String.split()`

Cross-references:

O'Reilly *JavaScript Definitive Guide* –page –49

Relational expression (Definition)

Relational expressions yield a Boolean result.

Availability:	ECMAScript edition –2
Property/method value type:	`Boolean primitive`

Relational expressions yield a Boolean result according to the relational test of the values either side of the operator.

Numerical values are compared according to sign and magnitude while string values are compared according to the Unicode collating sort sequence.

Relational operators will attempt to convert both arguments to a Number, and if at least one argument can be converted to a Number then the other will be forced to be a Number for comparison purposes. If both arguments are Strings or string-like objects, the relational test will be String-based.

If a prefixing plus sign is present, then a numeric coercion of a string takes place before the comparison.

Example code:

```
// Force a string comparison
myResult = (a+'' <= b+'');
// Force a numeric comparison
myResult = (a-0 <= b-0);
// Force a boolean comparison
myResult = (!a <= !b);
```

See also:	Equal to (==), Equality expression, Expression, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==), Relational operator, Type conversion

Cross-references:

ECMA 262 edition 2 –section –11.8

ECMA 262 edition 3 –section –11.8

Wrox *Instant JavaScript* –page –37

Wrox *Instant JavaScript* –page –39

Relational operator (Definition)

Relational operators are used to create relational expressions.

Availability:	ECMAScript edition –2
Property/method value type:	`Boolean primitive`

Relational operators are used in relational expressions and always yield a Boolean result.

Although generally considered to be members of the relational operator set, equality and non-equality tests are classified as equality operators in the ECMA standard.

The following table lists all operators that could be loosely classified or specifically classified as relational operators:

Operator	Description
`==`	Equal to
`===`	Identically equal to
`!=`	NOT equal to
`!==`	NOT identically equal to
`<`	Less than
`<=`	Less than or equal to
`>`	Greater than
`>=`	Greater than or equal to

It follows that all equality operators are generally classifiable as relational operators and all relational operators are members of the set of logical operators, since they all yield a Boolean value as a result.

See also:	Equal to (==), Equality operator, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==), Relational expression

Cross-references:

ECMA 262 edition 2 –section –11.8

ECMA 262 edition 3 –section –11.8

Wrox *Instant JavaScript* –page –19

releaseEvents() (Function)

Netscape Navigator 4 event management function.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myWindow*`.releaseEvents(`*anEventMask*`)`
	N	`releaseEvents(`*anEventMask*`)`
Argument list:	*anEventMask*	A mask defined with the manifest event constants

See also:	`captureEvents()`, `Document.captureEvents()`, `Document.releaseEvents()`, `Element.onevent`, Event handler, Event management, Event propagation, Event type constants, `Event.modifiers`, `Frame` object, `handleEvent()`, `Layer.captureEvents()`, `Layer.releaseEvents()`, `onLoseCapture`, `onMouseMove`, `Window` object, `Window.routeEvent()`, `Window.releaseEvents()`

Remainder (%) (Operator/multiplicative)

Divides one operand by another and yields the remainder.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*anOperand1* `%` *anOperand2*
Argument list:	*anOperand1*	The dividend value
	anOperand2	The divisor value

The `%` operator yields the remainder after dividing the left operand by the right. This is otherwise known as the modulo or modulus operator.

In an ECMAScript compliant interpreter, the remainder is a floating-point value and, unlike the C and C++ languages, the input operands can also be floating point values.

The result of performing a remainder on floating point values is not the same as the IEEE 754 remainder operation. IEEE 754 mandates a remainder computed via a rounding division whereas ECMA mandates a truncating remainder. In ECMAScript the remainder behaves similarly to the Java integer remainder operator and is analogous to the `fmod()` library function in C language compilers.

The behavior in an ECMA complaint JavaScript implementation should obey these rules:

- If either operand is `NaN` then the result is `NaN`.
- The sign of the result is the same as the sign of the dividend (the operand on the left).
- If the dividend is an infinity, the result is `NaN`.
- If the divisor is zero, the result is `NaN`.
- If the dividend is a finite value and the divisor is infinity, the result equals the dividend.
- If the dividend is zero and the divisor is finite, the result is zero.

Otherwise, as long as neither an infinity, zero or `NaN` is involved, the floating remainder is calculated like this. The dividend `n` is divided by the divisor `d` to produce a quotient, `q`. The quotient is forced to be an integer and multiplied again by `d` and the result subtracted from `n` to yield the remainder. Thus:

```
q = n/d
r = n - (d * q)
```

The value `q` will be forced to be an integer although the resulting value `r` may not be.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

Warnings:

- JScript version 1.0 truncates floating-point values to integers before applying the remainder operator. This means that the expression `5.5 % 2.2` yields the value `1`.

See also:	Arithmetic operator, Associativity, Divide (`/`), Divide then assign (`/=`), Integer arithmetic, Integer-value-remainder, `Math.ceil()`, `Math.floor()`, Multiplicative expression, Multiplicative operator, Operator Precedence, Remainder then assign (`%=`)

Cross-references:

ECMA 262 edition 2 –section –11.5.3

ECMA 262 edition 2 –section –11.13

ECMA 262 edition 3 –section –11.5.3

Wrox *Instant JavaScript* –page –19

Remainder then assign (%=) (Operator/assignment)

Divide one operand by another, leaving the remainder in the first.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`anOperand1 %= anOperand2`*
Argument list:	*`anOperand1`*	The dividend value where the result is also assigned
	`anOperand2`	The divisor value

Divide the left operand by the right operand and assign the remainder to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 % anOperand2;
```

Although this is classified as an assignment operator it is really a compound of an assignment and a multiplicative operator.

The associativity is right to left.

Refer to the operator precedence topic for details of execution order.

The new value of `anOperand1` is returned as a result of the expression.

Warnings:

- The operand to the left of the operator must be an *`LValue`*. That is, it should be able to take an assignment and store the value.

See also:	Arithmetic operator, Assign value (=), Assignment expression, Assignment operator, Associativity, Integer arithmetic, Integer-value-remainder, `LValue`, `Math.ceil()`, `Math.floor()`, Multiplicative operator, Operator Precedence, Remainder (`%`)

Cross-references:

ECMA 262 edition 2 –section –11.13

ECMA 262 edition 3 –section –11.13

request object (Object/NES)

A server-side object maintained by NES for each `HTTP:` request.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
JavaScript syntax:	NES	`request`
Object properties:	`<input_name>, <urlExtension>, agent, imageX, imageY, ip, method, protocol`	

Whenever a user requests a page from a Netscape Enterprise Server, a `request` object is created to manage storage, methods and properties for that request. When the request is returned, the object can be destroyed as it will have no further use.

The request object contains details of the URL, form data and search criteria. This is the way that you access the data in a form submit for example.

There are other properties belonging to this object that can tell you a lot about the client and what is happening there.

The ASP server also supports a `Request` object, whose name is capitalized. It's there for the same purpose and manages the incoming requests from the the clients' browsers.

Warnings:

- The Request object supported by ASP is quite different to that supported by NES. Since the tag introducer is quite different for each server-side system, it is unlikely you'll deploy common scripts across NES and ASP running on IIS.

See also:	Netscape Enterprise Server, `response.client`, `response.request`, `unwatch()`, `watch()`

Property	JavaScript	JScript	NES	Notes
<input_name>	1.1 +	-	2.0 +	-
<urlExtension>	1.1 +	-	2.0 +	-
agent	1.1 +	-	2.0 +	-
imageX	1.1 +	-	2.0 +	-
imageY	1.1 +	-	2.0 +	-
ip	1.1 +	-	2.0 +	Warning
method	1.1 +	-	2.0 +	-
protocol	1.1 +	-	2.0 +	-

request.<input_name> (Property)

Input elements can be accessed associatively by name.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	String primitive	
JavaScript syntax:	NES	request.*anInputElement*
Argument list:	*anInputElement*	The value of the NAME="..." tag attribute

The NAME="..." attribute of an <INPUT> element in a <FORM> can be used as a key to access the item as a property of the request object.

For example, a text filed can be created like this:

```
<INPUT NAME="MyNamedTextObject" TYPE="TEXT">
```

You can then access it as a property belonging to the response with this object and property reference:

```
response.MyNamedTextObject
```

request.<urlExtension> (Property)

Additional properties can be passed from URL extensions.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	String primitive	
JavaScript syntax:	NES	request.<urlExtension>

When you submit a request, you can pass parameter values in the request with a special coding technique that allows them to convey information in the URL that is reflected into special properties in the request object.

The question mark (?) is the control sequence introducer for this. Sometimes these values are referred to as the query or search portion of the URL when they are accessed as properties of an Anchor, Url, or Area object in the client.

The properties are then added as a name=value pair with multiple name=value pairs separated by an ampersand (&) character.

See also:	Anchor.search, Url.search

request.agent (Property)

A string containing the user agent details.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	String primitive	
JavaScript syntax:	NES	request.agent

This property contains the name and version of the client software making the request. This is a means of building client user agent dependent responses that deliver pages that are tailored to the particular browser being used.

You might use this to respond with a Netscape Navigator version of a page that behaves differently to the page you return to a user who requested the page from an MSIE browser.

request.imageX (Property)

The X coordinate of the mouse when an image map is clicked on.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	Number primitive	
JavaScript syntax:	NES	request.imageX

If a client-side input object whose type is "image" was used, this is the horizontal coordinate value of the mouse relative to the origin of the image when the user clicked the button on the image.

This would have been placed into the page with a tag like this:

```
<INPUT TYPE="image" NAME="imageName" SRC="...">
```

request.imageY (Property)

The Y coordinate of the mouse when an image map is clicked on.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	`request.imageY`

If a client-side input object whose type is "`image`" was used, this is the vertical coordinate value of the mouse relative to the origin of the image when the user clicked the button on the image.

This would have been placed into the page with a tag like this:

```
<INPUT TYPE="image" NAME="imageName" SRC="...">
```

request.ip (Property)

The IP address of the client.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`request.ip`

IP addresses are interesting and useful if you know their provenance but should not be relied upon to uniquely identify a user.

Warnings:

- Be careful when using this value. Proxy servers and firewalls can modify the value you actually see and can completely mask the IP address of the client. This can make all users behind the firewall or proxy appear to be using the same machine.
- Multiple users running Netscape Navigator via an X-Windows environment may indeed all be running the browser core on the same machine and only viewing the windows on their desktop systems.
- Finally, clients connected via an ISP may have floating IP address values and may be served through proxy and firewall systems. Indeed, some ISP-based users may even submit each request during a session through a different firewall. The session state can then only be identified by means of hidden values in the URL or with a cookie.

request.method (Property)

The request method determines to some extent how a server should respond to the request.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`request.method`

The request method could be one of the following:

- `get`
- `post`
- `head`
- `put`
- `delete`
- `options`
- `trace`

These values may be in upper or lower case so you should use a case-insensitive test if you need to check them.

request.protocol (Property)

The client may not support all the available protocols. This is the level of HTTP protocol that the client is prepared to accept.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`request.protocol`

HTTP/1.1 protocol is much more efficient than HTTP/1.0 because it reduces the number of TCP connections required. However, not all browsers support it and you may need to check this value if you are sending back multiple items in a single response.

Request-response loop (Definition)

The mechanism by which a web server handles a request and serves a page back to a browser.

The web browser sends a request to a web server. This is passed to the middleware application server which may create a special object to represent the request. Values passed to the web server by the browser will be reflected as properties of that request object.

This mechanism will invoke an appropriate handler. That handler may be compiled code or perhaps a script, possibly written in JavaScript in some middleware implementations.

As the handler is invoked, it is passed the request object and also an empty response object. The handler then populates the response object with headers and body contents. This may be done by means of property accessors or method calls.

Eventually the response is despatched back to the requesting browser. At this point some clean-up takes place, the request logging takes place and the request response loop is completed.

See also:	Storage duration

Requesting privileges (Security related)

Your script needs to request privileges when it requires them.

So long as you can sign scripts for Netscape Navigator, you can then make requests for privileges when those scripts run. This is done by means of the privilege manager.

Because the Netscape Navigator security model is based on the Java security model, the Netscape Navigator browser requests its privileges through the Java mechanisms. These are encapsulated in a class that you can access from inside JavaScript.

The downside of this is that there is no meaningful value returned when the request is made. If the request for a privilege is denied, the error causes a Java exception which is difficult to trap from JavaScript. It is possible that new browser versions will support an exception handling mechanism.

There are two principle methods that are useful here, one to request the privilege and the other to relinquish it.

`enablePrivilege()` –Requests the privilege passed as a string argument

`disablePrivilege()` –Relinquishes the privilege based on a string argument

Example code:

```
// Request the file reading
privilegenetscape.security.PrivilegeManager.enablePrivilege("UniversalFileRead")
```

See also:	`netscape.security.PrivilegeManager`, `PrivilegeManager` object, `PrivilegeManager.disablePrivilege()`, `PrivilegeManager.enablePrivilege()`, Signed scripts, `UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`, `UniversalFileRead`, `UniversalPreferencesRead`, `UniversalPreferencesWrite`, `UniversalSendMail`

Reserved Word (Definition)

JavaScript reserves certain words and maps computational functionality to them.

Availability:	ECMAScript edition –2

Reserved words are keywords that the interpreter uses to determine the instructions your script is going to execute.

Generally, reserved words will be one of the following:

- A keyword
- A reserved word
- A `null` literal
- A Boolean literal

Here is a complete list of reserved words as defined by the ECMA 262 standard. It also includes words that are dangerous because they are properties or method names.

Name	Notes
abstract	Java keyword reserved
alert	Identifier name
arguments	Identifier name
Array	Object type
blur	Identifier name
boolean	Java keyword reserved
Boolean	Object type
break	Keyword
byte	Java keyword reserved
callee	Identifier name
caller	Identifier name
captureEvents	Identifier name
case	Keyword
catch	Reserved word
char	Java keyword reserved
class	Reserved word
clearInterval	Identifier name
clearTimeout	Identifier name
close	Identifier name
closed	Identifier name

Table continued on following page

Name	Notes
confirm	Identifier name
const	Reserved word
constructor	Identifier name
continue	Keyword
Date	Object type
debugger	Reserved word
default	Keyword
defaultStatus	Identifier name
delete	Keyword
do	Keyword
document	Identifier name
double	Java keyword reserved
else	Keyword
enum	Reserved word
escape	Identifier name
eval	Identifier name
export	Keyword
extends	Reserved word
false	Boolean literal
final	Java keyword reserved
finally	Keyword
find	Identifier name
float	Java keyword reserved
focus	Identifier name
for	Keyword
frames	Identifier name
function	Keyword
Function	Object type
goto	Java keyword reserved
history	Identifier name
home	Identifier name
if	Keyword
implements	Java keyword reserved
import	Keyword
in	Keyword
Infinity	Global special variable
innerHeight	Identifier name
innerWidth	Identifier name

Table continued on following page

Name	Notes
instanceof	Keyword
int	Java keyword reserved
interface	Java keyword reserved
isFinite	Identifier name
isNaN	Identifier name
java	Identifier name
length	Identifier name
location	Identifier name
locationbar	Identifier name
long	Java keyword reserved
Math	Object type
menubar	Identifier name
moveBy	Identifier name
moveTo	Identifier name
name	Identifier name
NaN	Global special variable
native	Java keyword reserved
netscape	Identifier name
new	Keyword
null	Null literal
Number	Object type
Object	Object type
open	Identifier name
opener	Identifier name
outerHeight	Identifier name
outerWidth	Identifier name
package	Java keyword reserved
Packages	Identifier name
pageXOffset	Identifier name
pageYOffset	Identifier name
parent	Identifier name
parseFloat	Identifier name
parseInt	Identifier name
personalbar	Identifier name
print	Identifier name
private	Java keyword reserved
prompt	Identifier name

Table continued on following page

Name	Notes
protected	Java keyword reserved
prototype	Identifier name
public	Java keyword reserved
RegExp	Identifier name
releaseEvents	Identifier name
resizeBy	Identifier name
resizeTo	Identifier name
return	Keyword
routeEvent	Identifier name
scroll	Identifier name
scrollbars	Identifier name
scrollBy	Identifier name
scrollTo	Identifier name
self	Identifier name
setInterval	Identifier name
setTimeout	Identifier name
short	Java keyword reserved
static	Java keyword reserved
status	Identifier name
statusbar	Identifier name
stop	Identifier name
String	Object type
super	Reserved word
switch	Keyword
synchronized	Java keyword reserved
this	Keyword
throw	Keyword
throws	Java keyword reserved
toolbar	Identifier name
top	Identifier name
toString	Identifier name
transient	Java keyword reserved
true	Boolean literal
try	Keyword
typeof	Keyword
unescape	Identifier name
unwatch	Identifier name
valueOf	Identifier name

Table continued on following page

Name	Notes
var	Keyword
void	Keyword
volatile	Reserved word
watch	Identifier name
while	Keyword
window	Identifier name
with	Keyword

Looking at the some of the particular keywords being reserved, it suggests that future revisions of ECMAScript will become more object oriented. Or at least even if the underlying implementation is not truly object oriented, then the visible interface to the interpreter from a script will behave very much as if it is.

The third edition of the ECMA standard does not add any new keywords but removes those that have been defined as part of the upgrade to the standard. There are still several that are classed as reserved words that have already been implemented in some browser versions. Simply avoiding parse errors does not qualify them as features to be documented as being available in the browser.

On the Netscape developer web site, there is talk of developing a version 2.0 of the JavaScript language, to change JavaScript into a truly class-based object oriented language. At that time, many of the currently reserved words will likely become functional parts of the language.

Warnings:

- You must not use reserved words as identifier names.
- The table lists words you should avoid in a client-side context. You could argue that because `alert` is not a reserved word on the server-side that it would be safe to use it there. I would recommend against that because it is possible you might develop a library of JavaScript functions for general purpose use. It wouldn't be hard to imagine how an identifier that was server-side safe could eventually end up being executed client-side.
- The list in the table is not exhaustive and you should always think carefully about the identifier names you choose for properties, methods, functions, and variables.

See also:	`goto`, Label, Token

Cross-references:

ECMA 262 edition 2 –section –7.4.1

ECMA 262 edition 3 –section –7.5.1

O'Reilly *JavaScript Definitive Guide* –page –31

ResetButton object (Object/DOM)

A button in a form that will reset the form fields to their default values.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0
Inherits from:	`Input` object
JavaScript syntax:	- *myResetButton* = *myDocument*.*aFormName*.*anElementName* - *myResetButton* = *myDocument*.*aFormName*.elements[*anItemIndex*] IE *myResetButton* = *myDocument*.all.*anElementID* IE *myResetButton* = *myDocument*.all.tags("INPUT")[*anIndex*] IE *myResetButton* = *myDocument*.all[*aName*] - *myResetButton* = *myDocument*.forms[*aFormIndex*].*anElementName* - *myResetButton* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*] - *myResetButton* = *myDocument*.getElementById(*anElementID*) - *myResetButton* = *myDocument*.getElementsByName(*aName*)[*anIndex*] - *myResetButton* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
HTML syntax:	`<INPUT TYPE="reset">`
Argument list:	*anIndex* – A valid reference to an item in the collection *aName* – The `name` attribute of an element *anElementID* – The `ID` attribute of an element *anItemIndex* – A valid reference to an item in the collection *aFormIndex* – A reference to a particular form in the forms collection
Object properties:	`type, value`
Object methods:	`handleEvent()`
Event handlers:	`onAfterUpdate, onBeforeUpdate, onBlur, onClick, onDblClick, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onRowEnter, onRowExit`

Many properties, methods and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

There isn't really a `ResetButton` object class, but it is helpful when trying to understand the wide variety of input element types if we can reduce the complexity by discussing only the properties and methods of a reset button. In actual fact, the object is represented as an item of the `Input` object class.

Event handling support via properties containing function objects was added to `ResetButton` objects at version 1.1 of JavaScript.

Unlike MSIE, Netscape Navigator does not support the `defaultValue` property or the `select()` method in this sub-class of the `Input` object.

See also:	`Element` object, `Form.elements[]`, `FormElement` object, `Input` object, `Input.accessKey`, `onClick`, `ResetButton.handleEvent()`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onClick`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onRowEnter`	-	3.0 +	-	4.0 +	-	-	-	-
`onRowExit`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

ResetButton.handleEvent() (Method)

Passes an event to the appropriate handler for this object.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myResetButton`*`.handleEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An event to be handled by this object

This applies to Netscape Navigator prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`handleEvent(),ResetButton` object

ResetButton.type (Property)

The type value for the `<INPUT>` object that describes the reset button.

Availability:	DOM level –1 JavaScript –1.1 JScript –3.0 Internet Explorer –4.0 Netscape –3.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myResetButton.type`

The type value for a `ResetButton` is always "`reset`". This value is necessary to determine the type of form element because this object is really an instance of the `Input` class and not the `ResetButton` class. There is actually no `ResetButton` class.

See also:	`Input.type`

Property attributes:

`ReadOnly`.

ResetButton.value (Property)

The text string in the button.

Availability:	DOM level –1 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Opera –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myResetButton*.value

Although this value may be sent back to the web server when the form is submitted, the main purpose of a `ResetButton` is to reset the form element contents to the values defined in the HTML document source. This property provides a convenient means of labelling the button.

Warnings:

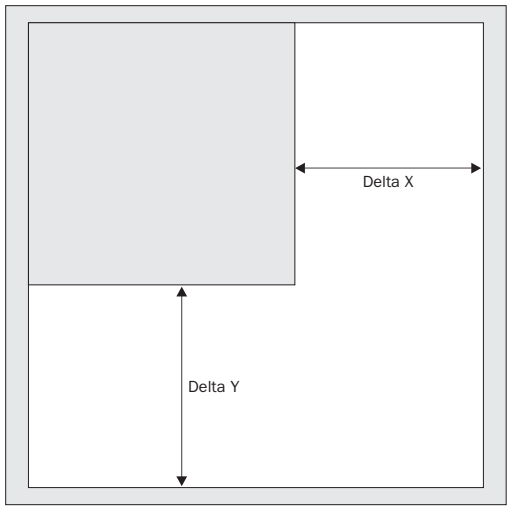
- This may be changed on some platforms but not others.

See also:	`Input.value`, `SubmitButton.value`

resizeBy() (Method)

An alias for the `window.resizeBy()` method.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*.resizeBy(*aChangeX*, *aChangeY*)
	-	resizeBy(*aChangeX*, *aChangeY*)
Argument list:	*aChangeX*	The difference in pixels
	aChangeY	The difference in pixels

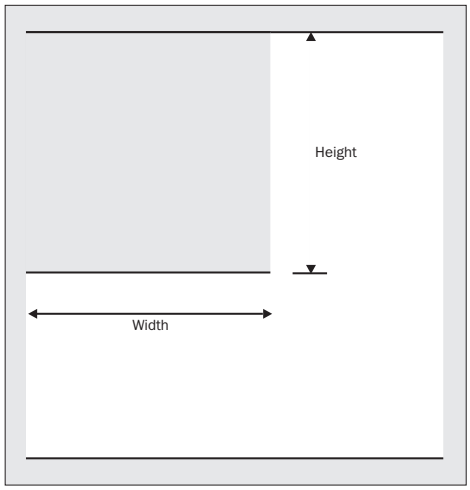


See also:	`Window.onresize`,`Window.resizeBy()`

resizeTo() (Method)

An alias for the `window.resizeTo()` method.

Availability:	JavaScript –1.2 JScript –3.0 Internet Explorer –4.0 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*`myWindow`*`.resizeTo(`*`aSizeX`*`, `*`aSizeY`*`)`
	-	`resizeTo(`*`aSizeX`*`, `*`aSizeY`*`)`
Argument list:	*`aSizeX`*	A distance in pixels
	`aSizeY`	A distance in pixels



See also:	`Window.onresize`,`Window.resizeTo()`

response object (Object/NES)

Part of the server-side support for JavaScript. This is the `Global` object in NES.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
JavaScript syntax:	NES	`response`
Object properties:	`client, database, project, request, server`	
Object methods:	`addClient(), addResponseHeader(), blob(), callC(), debug(), deleteResponseHeader(), flush(), getOptionValue(), getOptionValueCount(), redirect(), registerCFunction(), ssjs_generateClientID(), ssjs_getCGIVariable(), ssjs_getClientID(), trace(), write()`	

The response object in NES is the global object as well. This means the methods and properties that belong to the global object also belong to the response.

Because it is the global object there is no explicit way of referring to it.

We have collated all the response object methods and properties together in a group to aid the rapid location of related topics. Additional index entries are provided for the properties and methods as they would be addressed as members of the global object.

Warnings:

- The `Response` object is the `Global` object in a NES request-response context.

See also:	Netscape Enterprise Server, `unwatch()`, `watch()`

Property	JavaScript	JScript	NES	Notes
`client`	1.1 +	-	2.0 +	-
`database`	1.1 +	-	2.0 +	-
`project`	1.1 +	-	2.0 +	-
`request`	1.1 +	-	2.0 +	-
`server`	1.1 +	-	2.0 +	-

Method	JavaScript	JScript	NES	Notes
`addClient()`	1.1 +	-	2.0 +	-
`addResponseHeader()`	1.2 +	-	3.0 +	-
`blob()`	1.1 +	-	2.0 +	-
`callC()`	1.1 +	-	2.0 +	-
`debug()`	1.1 +	-	2.0 +	-
`deleteResponseHeader()`	1.2 +	-	3.0 +	-
`flush()`	1.1 +	-	2.0 +	-
`getOptionValue()`	1.1 +	-	2.0 +	-
`getOptionValueCount()`	1.1 +	-	2.0 +	-
`redirect()`	1.1 +	-	2.0 +	-
`registerCFunction()`	1.1 +	-	2.0 +	-
`ssjs_generateClientID()`	1.2 +	-	3.0 +	-
`ssjs_getCGIVariable()`	1.2 +	-	3.0 +	-
`ssjs_getClientID()`	1.2 +	-	3.0 +	-
`trace()`	1.1 +	-	2.0 +	-
`write()`	1.2 +	-	3.0 +	-

response.addClient() (Method)

Information about the client is added to a URL.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
JavaScript syntax:	NES	`addClient()`

Use this method to maintain state across requests by hiding information about the client in the URL. This allows you to send a page to the client and when they submit a form you can associate that request with the previous one.

See also:	Netscape Enterprise Server

response.addResponseHeader() (Method)

Adds a header record to the response object.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
JavaScript syntax:	NES	`addResponseHeader()`

The header to be added is presented as a name-value pair.

See also:	Netscape Enterprise Server

response.blob() (Method)

Extracts a binary large object from a data file in the server's file system.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`blob object`	
JavaScript syntax:	NES	`blob()`

BLOBs can be pulled out of the database and presented as images or as links (that is, documents).

See also:	Netscape Enterprise Server

response.callC() (Method)

Calls a native function within the server.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`callC(`*`aFunctionName`*`, `*`arguments`*`, ...)`
Argument list:	*`aFunctionName`*	The name of a function that has been registered
	`arguments`	A set of arguments to be passed to it when called

External library functions can be registered with the Netscape Enterprise Server and can then be called from scripts running in the request-response loop.

Such registered functions are invoked with the `callC()` method.

The function name is the one that was used when it was registered. Registering a function creates a wrapper that will always return a string primitive value.

See also:	Netscape Enterprise Server

response.client (Property)

A reference to a client object when scripts are used in NES.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`client object`	
JavaScript syntax:	NES	`client`

This global property points at a client object that is created automatically. The transactions between the Netscape Enterprise Server and the client browser maintain sufficient session state information that this object can be made available between one page request and the next.

See also:	`client` object, Netscape Enterprise Server, `request` object

response.database (Property)

A property that points at a globally available database access object.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`database object`	
JavaScript syntax:	NES	`database`

The object referred to by this property can be used to create connections to whatever database you have available in the server back-end environment.

See also:	`database` object

response.debug() (Method)

Prints a debugging message or value in the trace window.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
JavaScript syntax:	NES	`debug()`

This is a server-side method to assist in debugging script execution when developing scripts for NES.

This provides a means of watching the progress of a request-response handler which is running in a context that has no terminal I/O capabilities.

See also:	Netscape Enterprise Server

response.deleteResponseHeader() (Method)

Removes a specified header record from the response.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
JavaScript syntax:	NES	`deleteResponseHeader(`*`aHeader`*`)`
Argument list:	*`aHeader`*	The name of the header to be removed

This is part of the management of the request-response loop in the server. Many response headers will be added automatically by the NES server, and you may want to add others. Occasionally, the server may place a header into the response that you don't want to send back to the user. This method can be used to remove such a header.

See also:	Netscape Enterprise Server

response.flush() (Method)

Sends the current contents of the output buffer to the client.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
JavaScript syntax:	NES	`flush()`

Response writing tends to involve a certain amount of buffering by the response manager in the server back-end. This is necessary to improve performance and throughput. The data is only physically transferred to the client when a buffer full of data is ready or when the response is completed. This can mean that a JavaScript error can leave the response incomplete if it fails in a way that prevents the response from being completed properly. A run-time error would normally not write any pending response data out to a client or it might result in part of the data being sent but probably not all.

The `flush()` method allows you to force the response output to the client to be updated so that the response is complete up to this point and there are no pending contents yet to be transmitted. You might force a `flush()` at the end of a database record being read and processed for example.

See also:	`File.flush()`, Netscape Enterprise Server

response.getOptionValue() (Method)

Returns the value of the selected option items in a `<SELECT>` block.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`getOptionValue()`

Server-side access to the `<OPTION>` tag contents in a `<SELECT>` block is accomplished slightly differently in an NES server than in a Netscape Navigator browser. This is because the viewpoint is different as you look at the DOM. The DOM standard relates predominantly to how the document is modelled in the client.

See also:	Netscape Enterprise Server, `Option` object, `Select` object

response.getOptionValueCount() (Method)

Returns the number of items in a `<SELECT>` block.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	`getOptionValueCount()`

This would normally be the `length` property of a collection or array.

See also:	Netscape Enterprise Server, `Option` object, `Select` object

response.project (Property)

A reference to a `project` object when scripts are executed in NES.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`project object`	
JavaScript syntax:	NES	`project`

This global property points at a `project` object that is created when the server application is starts running. The `project` object is created automatically and a reference to it is stored here for use when needed.

See also:	Netscape Enterprise Server, `project` object

response.redirect() (Method)

Send the appropriate headers and meta information to cause the client to do a redirect to a different URL.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
JavaScript syntax:	NES	`redirect(`*`URL`*`)`

This is a convenience method for populating headers in the response.

See also:	Netscape Enterprise Server

response.registerCFunction() (Method)

Register a native C language function for use with `callC()`.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	NES	`registerCFunction(`*`aFunctionName`*`, `*`aLibrary`*`, `*`aCFunction`*`)`
Argument list:	*`aCFunction`*	The name of the C function in the library
	`aFunctionName`	The name of the function in the JavaScript environment
	`aLibrary`	A path to the library containing the C function

External library functions can be registered with the Netscape Enterprise Server and can then be called from scripts running in the request-response loop.

Such registered functions are invoked with the `callC()` method.

The registration process creates a mapping between the JavaScript environment and the C language environment. This suggests that the two need not be named the same although it is good practice to preserve the names recognizably across such wrapping mechanisms.

The C function is encapsulated in a JavaScript object wrapper.

A Boolean `true` value is returned if the registration succeeds and a Boolean `false` if it fails.

See also:	Netscape Enterprise Server

response.request (Property)

A reference to a `request` object that encapsulates a client request arriving in an NES server.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`request object`	
JavaScript syntax:	NES	`request`

This global property points at a `request` object that is created when the client submits a request to the server. The `request` object is created automatically and a reference to it is stored here for use when needed.

See also:	Netscape Enterprise Server, `request` object

response.server (Property)

A reference to the `server` object is created automatically when the server is started.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
Property/method value type:	`server object`	
JavaScript syntax:	NES	`server`

This property refers to the globally available server-wide shared `server` object. There is only one of these and it is available to all sessions, application projects and client requests.

This is an object that allows you to share property values across all sessions running in all applications across the entire server. The locking facilities permit you to lock resources while you are using them.

Because this applies server-wide, there is even more reason to ensure you lock objects for the minimum of time and relinquish the locks as soon as possible. It is quite feasible to completely stall the whole server by locking a vital resource during the processing of a single client request. The effect of this is to make your server a single-threaded non-concurrent session server. That is, it will only actually serve one client request at a time.

See also:	Netscape Enterprise Server, server object

response.ssjs_generateClientID() (Method)

Generate a unique identifier for a new `client` object.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`ssjs_generateClientID()`

There may be times during session-state management that you need a unique `ID` value for the clients. This method provides a means of generating a guaranteed unique `ID` value.

See also:	Netscape Enterprise Server

response.ssjs_getCGIVariable() (Method)

Return the value of the requested environment variable.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`ssjs_getCGIVariable()`

The server-side execution occurs within an environment that may have been inherited from the hosting operating system. Other environment values will have been provided by the web server.

See also:	Netscape Enterprise Server

response.ssjs_getClientID() (Method)

Obtains the unique identifier value from a `client` object.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`ssjs_getClientID()`

This is a somewhat inconsistent way of obtaining property values and there may be other undocumented ways of obtaining client `ID` values.

See also:	Netscape Enterprise Server

response.trace() (Method)

A server-side method to assist in tracing script execution when developing scripts for NES.

Availability:	JavaScript –1.1 Netscape Enterprise Server –2.0	
JavaScript syntax:	NES	`trace()`

Debugging server-side execution is difficult because you cannot normally see the execution happening. Placing a debugging console onto the server is notoriously difficult, so this method may provide some much needed assistance when trying to debug a non-working request handler.

See also:	Netscape Enterprise Server

response.write() (Method)

Writes the string value passed as an argument to the outgoing HTML response stream.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
JavaScript syntax:	NES	`write()`

This is a server-side method that is not to be confused with the method that belongs to the `document` object.

The `write()` method can be used to output values generated by any legal JavaScript. It is the only way that server-side JavaScript can generate any output and it should be used to generate HTML for insertion into the document.

You can get some quite interesting output if you are prepared to study and exploit the server-side environment.

See also:	`Document.write()`,`Document.writeln()`, Netscape Enterprise Server

Restricted access (Definition)

There are ways in which security is restricted and controlled by means of privileges.

The private information in the `history` object can only be accessed if a script has `UniversalBrowserRead` privileges.

See also:	Data-tainting, Security policy, `UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`, `UniversalFileRead`, `UniversalPreferencesRead`, `UniversalPreferencesWrite`, `UniversalSendMail`

ResultSet object (Object/NES)

This is part of the database access suite in Netscape Enterprise Server. It is returned by a stored procedure call.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
JavaScript syntax:	nes	*myResultSet* = *myStProc*.resultSet();
Object properties:	`prototype`	
Object methods:	`close(), columnName(), columns(), next()`	

When you call a stored procedure in a RDMS, you don't always get back a sequence of records in the same layout and structure as when you just do a simple SQL select style query.

An SQL query would return a series of records separated by newline characters. A stored procedure might return a mixed collection of records of different types.

A `ResultSet` object is created by asking the `Stproc` object for it when the stored procedure has been called and returned from the database.

The traversing mechanisms provided with a result set allow you to move forwards through the data but you cannot move backwards. Also, you can only read values from a result set as opposed to a `cursor` which allows you to update it and write new values back.

Example code:

```
<SERVER>
// An example derived from Wrox Professional JavaScript
database.connect("ODBC", "myDatabase", "me", "myPassword", "");
myStoredProc = database.storedProc("myProcedure", 40);
myResultSet = myStoredProc.resultSet();
</SERVER>
```

See also:	`database.storedProc()`, Netscape Enterprise Server, `Stproc.resultSet()`, `unwatch()`, `watch()`

Property	JavaScript	JScript	NES	Notes
`prototype`	1.2 +	-	3.0 +	-

Method	JavaScript	JScript	NES	Notes
`close()`	1.2 +	-	3.0 +	-
`columnName()`	1.2 +	-	3.0 +	-
`columns()`	1.2 +	-	3.0 +	-
`next()`	1.2 +	-	3.0 +	-

ResultSet.close() (Method)

Closes the `ResultSet` object when you have finished accessing its contents.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
JavaScript syntax:	NES	*`myResultSet`*`.close()`

Although closures generally get dealt with automatically for you when a request handler exits, it's good style to call the closure methods yourself when you no longer need the database connection.

See also:	`Stproc.close()`

ResultSet.columnName() (Method)

Returns the name of the column with the specified index number.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`myResultSet`*`.columnName(`*`anIndex`*`)`
Argument list:	*`anIndex`*	A valid column number

If you are enumerating through the columns in the result set, this yields up the name of the indexed column. You may need to use this technique if you have a table structure that could change.

ResultSet.columns() (Method)

Returns the number of columns in the result set.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*`myResultSet`*`.columns()`

If your tables are being modified in the database or you are changing the way the result set is requested, the number of columns may be indeterminate until you actually make the request. This method provides a column count so you can enumerate through the columns to extract the values from them.

See also:	`Stproc.outParamCount()`

ResultSet.next() (Method)

Moves the access pointer to the next row in the result set.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
JavaScript syntax:	NES	*`myResultSet`*`.next()`

When building tables of extracted database content, this provides a way to enumerate through the rows one at a time.

ResultSet.prototype (Property)

The prototype for the `ResultSet` object that can be used to extend the interface for all `ResultSet` objects.

Availability:	JavaScript –1.2 Netscape Enterprise Server –3.0	
Property/method value type:	`ResultSet` object	
JavaScript syntax:	NES	`ResultSet.prototype`
	NES	`myResultSet.constructor.prototype`

Refer to:

`prototype` property

return (Statement)

Returns control back to the caller of a function.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0	
JavaScript syntax:	-	`return(`*anExpression*`);`
	-	`return` *anExpression*`;`
	-	`return;`
Argument list:	*anExpression*	A value to return to the function caller

A `return` keyword is a jump statement. It is used to unconditionally exit from a function, pass back a result, and make execution flow to the caller of the function.

When the `return` statement is executed, the execution context is disposed of and removed from the stack. Execution continues at the point in the caller where the function was invoked. The function is replaced by the value being returned.

If the function is not being assigned to an LValue or Left Hand Side expression or has been cast to a `void` type, the result will be discarded.

If the expression is omitted in the `return` statement, the `undefined` value is returned in its place. While compiled languages are far more particular about the presence or absence of this expression, JavaScript is far more forgiving.

Functions that return undefined values are likely to be used as procedures rather than functions. A procedure is invoked as a statement that stands alone. The intent of a function is to return a result that will be substituted in its place.

It is considered illegal for the `return` statement to be present in any statement block other than that belonging to a function. However it can exist inside the statement block associated with a conditional statement or iterator statement as long as they themselves are within a function block. They may be nested more than one level deep but must ultimately belong to a function.

Warnings:

- It is considered to be a syntax error to use the `return` statement anywhere other than in a function body.
- You will not get the `return` value back properly if there is a line terminator between the `return` keyword and the value it was supposed to return. There is a temptation to break long strings over several lines like this:

```
return

"A very long string goes here ...";
```

- This will return the value `undefined` and not the string you intended to return. It is probably better style to assign the string to a variable and return that but there are implications there of string construction-destruction, garbage collection, and potential memory leaks and to trade those problems off it's best to try and eliminate string creation and memory usage if possible.

Example code:

```
// Declare a procedure with an implied return
function aProcedure()
{
document.write("Hello");
}
// Declare a procedure that returns an undefined value
function anotherProcedure()
{
   alert("Click OK to continue");
   return;
}
// Declare a function that returns a result
function aRealFunction()
{
   return 1000;
}
// Use the functions and procedures
aProcedure();
anotherProcedure();
x = aRealFunction();
```

See also:	`break`, Completion type, `continue`, `function( ... ) ...`, Iteration statement, Jump statement, Statement

Cross-references:

ECMA 262 edition 2 –section –12.9

ECMA 262 edition 3 –section –12.9

Wrox *Instant JavaScript* –page –27

returnValue (Property)

An alias for the `window.returnValue` property.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`User defined`	
JavaScript syntax:	IE	*myWindow*`.returnValue`
	IE	*myWindow*`.returnValue = `*aNewValue*
	IE	`returnValue`
	IE	`returnValue = `*aNewValue*
Argument list:	*aNewValue*	The value to be returned when the modal dialog closes

Refer to:

`Window.returnValue`

RevealTrans() (Filter/reveal)

A reveal filter for controlling transitions.

Availability:	JScript –3.0 Internet Explorer –4.0

Refer to:

filter – `RevealTrans()`

rgb() (Function)

A special color definition function used in style sheet color specifications.

Availability:	JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
JavaScript syntax:	-	`rgb(`*`rValue gValue bValue`*`)`
Argument list:	*`bValue`*	Blue intensity
	`gValue`	Green intensity
	`rValue`	Red intensity

The values are separated by spaces and you must specify all three. This value can then be used in the property assignment for a style property that controls color. You can use it in any position where a color value would be used.

See also:	Color value, `style.backgroundColor`, `style.borderColor`, `style.color`, `style.outlineColor`

Right shift (Operator/bitwise)

A rightwards shift of a bit pattern.

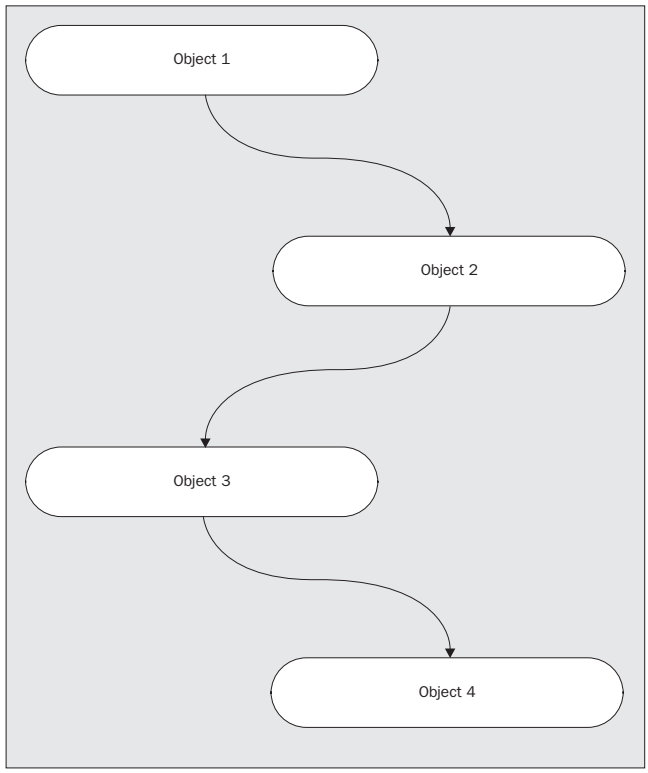
Refer to:

Bitwise shift right (>>)

routeEvent() (Function)

Part of the Netscape Navigator 4 event propagation complex.

Availability:	JavaScript –1.2 Netscape –4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myWindow*`.routeEvent(`*anEvent*`)`
	N	`routeEvent(`*anEvent*`)`
Argument list:	*anEvent*	An `event` object



See also:	`captureEvents()`, Event handler, Event management, `handleEvent()`, `Window.routeEvent()`

Cross-references:

Wrox *Instant JavaScript* –page –55

rows object (Definition)

Some documentation sources describes a `rows` object class that contains an object representing each row in a table. There is no such class, it's simply a collection.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0
Object properties:	`length`
Object methods:	`item(), tags()`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`length`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`item()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`tags()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Refer to:

`Collection` object

RT object (Object/HTML)

This is the ruby text associated with a `RUBY` object.

Availability:	JScript –5.0 Internet Explorer –5.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myRT* = *myDocument*.all.*anElementID*
	IE	*myRT* = *myDocument*.all.tags("RT")[*anIndex*]
	IE	*myRT* = *myDocument*.all[*aName*]
	-	*myRT* = *myDocument*.getElementById(*anElementID*)
	-	*myRT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	myRT = *myDocument*.getElementsByTagName("RT")[*anIndex*]
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Collections:	`all[], attributes[], childNodes[], children[], filters[]`	

See also:	RUBY object, style.rubyAlign, style.rubyOverhang, style.rubyPosition, Element object

Inheritance chain:

Element object, Node object

RUBY object (Object/HTML)

A ruby is an annotation or pronunciation guide for a string of text. The string of text annotated with a ruby is referred to as the base.

Availability:	JScript –5.0 Internet Explorer –5.0	
Inherits from:	Element object	
JavaScript syntax:	IE	*myRuby* = *myDocument*.all.*anElementID*
	IE	*myRuby* = *myDocument*.all.tags("RUBY")[*anIndex*]
	IE	*myRuby* = *myDocument*.all[*aName*]
	-	*myRuby* = *myDocument*.getElementById(*anElementID*)
	-	*myRuby* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myRuby* = *myDocument*.getElementsByTagName("RUBY")[*anIndex*]
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Collections:	all[], attributes[], childNodes[], children[], filters[]	

To create a RUBY object, you use the <RUBY> HTML tag like this:

```
<RUBY>

Some base text

<RT>Some ruby text

</RUBY>

The <RT> tag creates an RT object.
```

See also:	Element object, style.rubyAlign, style.rubyOverhang, style.rubyPosition

Inheritance chain:

`Element` object, `Node` object

rule object (Object/DOM)

An object that contains a single CSS styling rule.

Availability:	DOM level –2 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0		
JavaScript syntax:	IE	*myRule* = *myDocument*.all.aStyleSheetID.rules[*anIndex*]	
	IE	*myRule* = *myStyleSheet*.rules[*anIndex*]	
	IE	*myRule* = *mySelectorArray*[*anIndex*]	
	-	*myRule* = *myDocument*.styleSheets[*anIndex*].cssRules[*anIndex*]	
Argument list:	*anIndex*		A reference to an element in a collection
Object properties:	`cssText, parentStyleSheet, readOnly, runtimeStyle, selectorText, style`		

This is referred to as a selector and one or more declarations within a cascading style sheet (CSS). It is supported by MSIE.

DOM level 2 calls this a `CSSRule` object. It also describes a `CSSStyleRule` object as a sub-class of that object. The MSIE browser implements both as a single class. The `CSSRule` class maintains the following named constants:

Value	Name	DOM
0	UNKNOWN_RULE	2
1	STYLE_RULE	2
2	CHARSET_RULE	2
3	IMPORT_RULE	2
4	MEDIA_RULE	2
5	FONT_FACE_RULE	2
6	PAGE_RULE	2

DOM level 2 specifies these additional properties:

- `type`
- `parentRule`

See also:	`Document.styleSheets[]`, `SelectorArray` object, `StyleSheet` object, `StyleSheet.rules[]`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
cssText	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-
parentStyleSheet	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-
readOnly	-	3.0 +	-	3.0 +	-	-	ReadOnly
runtimeStyle	-	5.0 +	-	5.0 +	-	-	-
selectorText	1.5 +	3.0+	6.0 +	4.0+	-	2 +	ReadOnly
style	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-

rule.cssText (Property)

The CSS text belonging to a style sheet rule.

Availability:	DOM level –2 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myRule*`.cssText`

This is the text that would be entered into a style sheet to describe a particular CSS rule. Since it is returned as text, there is no implied structure to it so you will need to parse it out if necessary.

rule.parentStyleSheet (Property)

The style sheet that owns this `rule` object.

Availability:	DOM level –2 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0	
Property/method value type:	`StyleSheet object`	
JavaScript syntax:	-	*myRule*`.parentStyleSheet`

There is a hierarchy of objects contained in the CSS style sheet support. Rules describe a style and are collected together into owning parent style sheets.

See also:	Hierarchy of objects

rule.readOnly (Property)

Some rules can be set read-only to prevent their styles from being changed.

Availability:	JScript –3.0 Internet Explorer –4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*`myRule`*`.readOnly`

You may want to define a style in a style sheet and lock it down to prevent the JavaScript code from altering its appearance. Setting this property to `true` will lock the rule and prevent it from being changed.

See also:	StyleSheet.readOnly

Property attributes:

`ReadOnly`.

rule.runtimeStyle (Property)

The style values at run-time taking into account all cascades and dynamic style changes.

Availability:	JScript –5.0 Internet Explorer –5.0	
Property/method value type:	`Style` object	
JavaScript syntax:	IE	*`myRule`*`.runtimeStyle`

Various objects are used to describe the cascading style effect. Rules are associated with style sheets and yet more are instantiated and associated with objects. The current style takes into account the cascaded effect and run-time style further extends this to include style changes that are driven by scripts. This gives you a level of access that may help to un-cascade a style and apply new styling effects dynamically as needed.

rule.selectorText (Property)

The selector text for a rule.

Availability:	DOM level –2 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myRule`*`.selectorText`

This is the part of the CSS style definition text that associates the rule with a document element. For example, a rule that applies to the `BODY` of a document has the value "`BODY`" stored in its `selectorText` property.

See also:	`SelectorArray` object

Property attributes:

`ReadOnly`.

rule.style (Property)

The style settings for a rule.

Availability:	DOM level –2 JavaScript –1.5 JScript –5.0 Internet Explorer –5.0 Netscape –6.0
Property/method value type:	`Style` object
JavaScript syntax:	- *myRule*.style

The rule is the owner of a `style` object which contains properties governing the various style attributes. The order and priority of the rules dictate how the `style` objects cascade to create the `currentStyle` and the `runtimeStyle` objects.

See also:	`style` object (2)

runtimeStyle object (Object/JScript)

A style that applies to an object at run-time and overrides other style settings.

Availability:	JScript –5.0 Internet Explorer –5.0
Inherits from:	`style` object
JavaScript syntax:	IE *myRuntimeStyle* = *myElement*.runtimeStyle
Object methods:	`getAttribute(), getExpression(), removeExpression(), setAttribute(), setExpression()`

This represents the cascaded format and style of its parent object. The value in this object overrides global style-sheets, inline styles and HTML tag attribute values. It overwrites the values provided by `currentStyle` objects but not those supplied by the `style` object.

The properties belonging to this object correspond closely to those of the `style` object and so there is little point in discussing them again here. Refer to the `style` object property descriptions for details of the various properties.

Because the style values are cascaded from style sheet to style sheet and may include some inline styles as well as some explicit styles, objects need to maintain a current style value that is the result of all the inheritances applied on top of one another.

In addition they maintain a run-time style which reflects dynamic changes as well. The `runtimeStyle` is based on the `currentStyle` originally.

See also:	`currentStyle` object, `Element.currentStyle`, `Element.runtimeStyle`, `style` object (2)

Method	JavaScript	JScript	N	IE	Opera	Notes
`getAttribute()`	-	5.0 +	-	5.0 +	-	-
`getExpression()`	-	5.0 +	-	5.0 +	-	-
`removeExpression()`	-	5.0 +	-	5.0 +	-	-
`setAttribute()`	-	5.0 +	-	5.0 +	-	-
`setExpression()`	-	5.0 +	-	5.0 +	-	-

RValue (Definition)

The result of evaluating an expression.

An `RValue` is the value of an expression. This is based on the assignment expression in its general form being like this:

```
LValue = RValue;
```

This is otherwise known as the "value of an expression".

An `RValue` can be a variable or a constant whereas an LValue must be a variable or some other identifier that indicates a modifiable storage location.

See also:	Assignment expression, Definition, LValue, Property value

S object (Object/HTML)

An object that represents the font style controlled by the <S> HTML tag.

Availability:	JScript –3.0 Internet Explorer –4.0	
Deprecated:	Yes	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myS* = *myDocument*.all.anElementID
	IE	*myS* = *myDocument*.all.tags("S")[*anIndex*]
	IE	*myS* = *myDocument*.all[*aName*]
	-	*myS* = *myDocument*.getElementById(*anElementID*)
	-	*myS* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myS* = *myDocument*.getElementsByTagName("S")[*anIndex*]
HTML syntax:	<S> ... </S>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

Event name	JavaScript	JScript	Nav	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-

Table continued on following page

Event name	JavaScript	JScript	Nav	IE	Opera	DOM	HTML	Notes
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

Same origin (Security related)

A policy for granting access across window boundaries.

The same origin policy is a foundational concept as far as browser security is concerned. Put simply, it states that a script can access the contents of another window or frame if the `HREF` for that target was loaded from the same host at the same IP port number and with the same protocol.

This ensures that `http:` pages cannot read `https:` content and that pages served by a web server on port 80 cannot read values from a potentially different web server on port 8080 for example. Both of those also require that the host be the same.

This can be circumvented with `UniversalBrowserRead` privilege which allows properties to be read from windows containing objects that were from a different origin. The `UniversalBrowserWrite` property allows those objects with a different origin to be modified. Granting both would allow a script to read and write properties in a window with a different origin.

The same origin policy applies to most but not all properties of a window. It does apply to almost every property belonging to a document object.

You can allow documents from different origins to access properties belonging to your window and document but you need to provide a public API to let them do this. You can alias the private properties by publishing them as user-defined values.

You can also relax the same origin policy as far as hostnames are concerned by setting the domain property. You could set the domain value inside a document as long as it is a genuine fragment of the host name. If you do this in two documents, both served from different hosts belonging to a higher level domain that is the same, the same origin policy is relaxed when the domain value is identical for both documents.

Warnings:

- The common domain access relaxation technique only works for JavaScript version 1.1 and higher.

See also:	`export`, `import`, Security policy, Signed scripts, `UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`

SAMP object (Object/HTML)

An object representing the HTML content delimited by the `<SAMP>` tags.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*mySAMP* = *myDocument*.all.anElementID
	IE	*mySAMP* = myDocument.all.tags("SAMP")[*anIndex*]
	IE	*mySAMP* = *myDocument*.all[*aName*]
	-	*mySAMP* = *myDocument*.getElementById(*anElementID*)
	-	*mySAMP* = myDocument.getElementsByName(*aName*)[*anIndex*]
	-	*mySAMP* = *myDocument*.getElementsByTagName("SAMP")[*anIndex*]
HTML syntax:	`<SAMP> ... </SAMP>`	
Argument list:	*anElementID*	The `ID` value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	1.0 +	-	3.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	1.0 +	-	3.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

Scalar type (Definition)

Data types composed of a single arithmetic atomic component.

Scalar type values are those which are not aggregate and which are numeric or arithmetic in capability.

See also:	Aggregate type, Type

Scope (Definition)

The part of the script within which an identifier is reachable.

An identifier is scoped to be globally available everywhere or locally available only within a function block.

However, JavaScript allows identifier names in local contexts to override those in the global context.

This means that an identifier name can be reused within a function and the local value will be used in place of the global value. Other global variables not overridden will be accessed from the global pool.

Example code:

```
// Declare a global value
var myVariable = "Global Value";
// Declare a local value inside a function
 bodyfunction local_scope(){var myVariable = "Local Value";
document.write(myVariable);}
// Demonstrate the scope override local replacing global
document.write(myVariable);
local_scope();
```

See also:	`__parent__`, Scope chain, Variable

Cross-references:

Wrox *Instant JavaScript* – page – 27

Scope chain (Definition)

A scope chain is a logical list of objects associated with an execution context.

Availability:	ECMAScript edition – 2

Identifiers are accessible within the scope of the function body they are created in, or within the global scope.

A scope chain is a logical list of objects associated with an execution context and which indicates the order in which target objects are bound to identifiers.

When the flow of control enters an execution context, the scope chain is created and populated with an initial set of objects. What objects these are depends on the type of code being executed. The scope chain is destroyed on exit from the execution context. This means that factors that affect the construction and ordering of a scope chain may cause it to be created differently each time a particular execution context is entered.

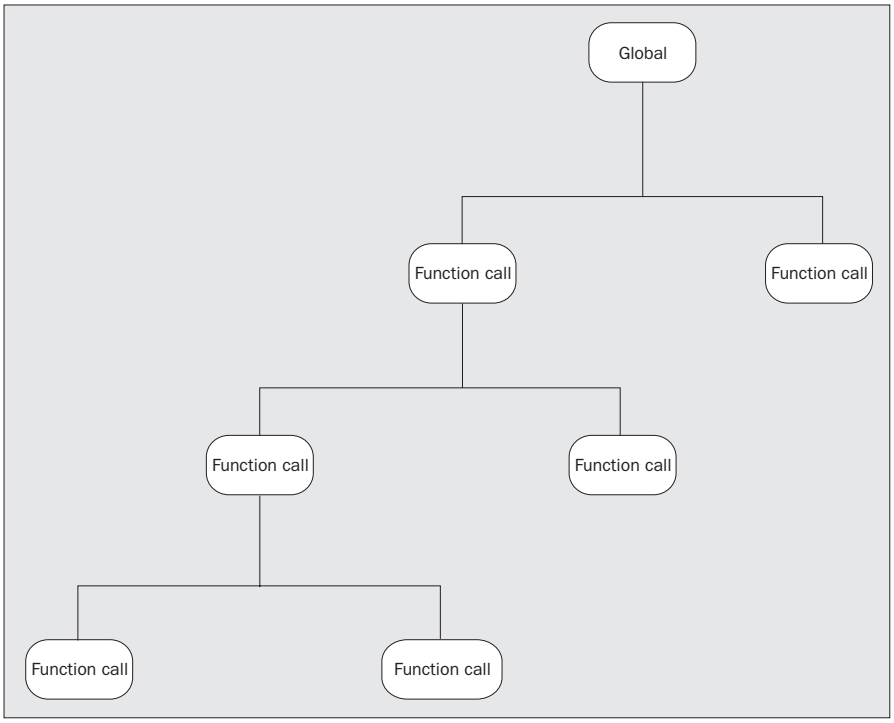
Calling one function from within another builds an increasingly deep scope chain containing each function's identifiers as it builds.

During execution, the scope chain of an execution context is only affected by the presence of a `with` code block. If a `with` block is entered, the scope chain has the `with` block added to the front of it. This is because variables may be local to the `with` block and may override earlier defined variables during the name binding process. When the `with` block is left, the scope chain is bumped to remove the item at the front. This happens when the code exits normally, with a `break` or because of a `continue` statement.

This scope chain behaviour is different to that you may be used to with a procedural language where the scope is block structured.

The scoping rules allow for identifiers in the global pool to be overridden by identifiers in the local context that have the same name.

Scope chains are only affected by `with` blocks and function calls. An `if` block or an iterator loop block does not have a context in which to create a new scope item. In a compiled language, such as ANSI C, you can create variables that are local to conditional and iterator blocks. The code in a conditional and iterator block in a JavaScript session uses the scope chain as it was when the block was entered.



See also:	`__parent__`, `break`, Call by value, Completion type, Compound statement, continue, Declaration, Execution context, Identifier, Identifier resolution, Implementation-supplied function, Namespace, Parameter, Scope, Storage duration, Variable, `with ...`

Cross-references:

ECMA 262 edition 2 – section – 10.1.1

ECMA 262 edition 2 – section – 10.1.4

ECMA 262 edition 3 – section – 10.1.4

Wrox *Instant JavaScript* – page – 27

Scope of event handler (Definition)

The scope of an event handler is somewhat different to the normal scope.

Refer to:

Event handler scope

screen (Property)

An alias for the `window.screen` property.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Screen object`	
JavaScript syntax:	-	*myWindow*`.screen`
	-	`screen`

Property attributes:

`ReadOnly`.

Refer to:

`Window.screen`

Screen object (Object/browser)

An object that represents the screen display and its rendering capabilities.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0	
JavaScript syntax:	-	*myScreen* `=` *myWindow*`.screen`
	-	*myScreen* `= screen`
Object properties:	`availHeight, availLeft, availTop, availWidth, bufferDepth, colorDepth, fontSmoothingEnabled, height, pixelDepth, updateInterval, width`	

The properties of this object describe the physical attributes of the display screen the browser is currently operating in. The values reflected by users of desktop computers will describe a much higher resolution than a WebTV set-top box, for example. Other set-top boxes for use with TV should conform to the same resolution but it is likely that they will all vary slightly from one another.

Warnings:

- Some documentation resources have referred to this object as a `screen` object rather than a `Screen` object. Note the capitalization. When examined, we found that several browsers spell the object with a capital S.

See also:	JellyScript, `Window`.screen

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
availHeight	1.2 +	3.0 +	4.0 +	4.0 +	5.0+	-	-	ReadOnly
availLeft	1.2 +	-	4.0 +	-	-	-	-	ReadOnly
availTop	1.2 +	-	4.0 +	-	-	-	-	ReadOnly
availWidth	1.2 +	3.0 +	4.0 +	4.0 +	5.0+	-	-	ReadOnly
bufferDepth	-	3.0 +	-	4.0 +	-	-	-	-
colorDepth	1.2 +	3.0 +	4.0 +	4.0 +	5.0+	-	-	ReadOnly
fontSmoothing Enabled	-	3.0 +	-	4.0 +	-	-	-	-
height	1.2 +	3.0 +	4.0 +	4.0 +	5.0+	-	-	ReadOnly
pixelDepth	1.2 +	-	4.0 +	-	5.0+	-	-	ReadOnly
updateInterval	-	3.0 +	-	4.0 +	-	-	-	-
width	1.2 +	3.0 +	4.0 +	4.0 +	5.0+	-	-	ReadOnly

Screen.availHeight (Property)

The available height taking task bars and menu bars into consideration.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myScreen*.availHeight

This property is extremely important when developing content for platforms with different screen sizes. Although historically, screen sizes have been more or less the same, with the advent of web on TV set-top boxes, this property can have a much wider range of possble values.

Property attributes:

`ReadOnly`.

Screen.availLeft (Property)

The left-most pixel that is accessible in this screen.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myScreen*.availLeft

Property attributes:

`ReadOnly`.

Refer to:

`Screen.availHeight`

Screen.availTop (Property)

The top-most pixel that is accessible in this screen.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*`myScreen`*`.availTop`

Property attributes:

`ReadOnly`.

Refer to:

`Screen.availHeight`

Screen.availWidth (Property)

The available width taking task bars and menu bars into account.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myScreen`*`.availWidth`

Property attributes:

`ReadOnly`.

Refer to:

`Screen.availHeight`

Screen.bufferDepth (Property)

The pixel Z-depth for the off-screen buffer.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	`myScreen.bufferDepth`

This property is important if you are trying to match colors properly. Knowing that you have a limited display palette available means you can optimize your content accordingly.

Screen.colorDepth (Property)

The number of bits available to resolve color values.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myScreen`*`.colorDepth`

To establish the number of available colours, raise the value 2 to the power of this value.

The technical description of this value is that it is the base-2 logarithm of the number of colours available.

You will likely get one of the following values:

- 1 bit – black and white
- 8 bits – 256 colours
- 16 bits – thousands of colours
- 24 bits – millions of colours
- 32 bits – millions of colours (possibly with transparency thrown in)

Other values are possible but unlikely.

Property attributes:

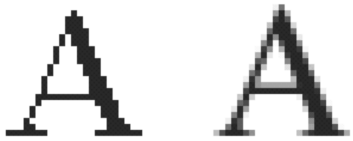
`ReadOnly`.

Screen.fontSmoothingEnabled (Property)

A switch to control the kind of font rendering on the screen.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myScreen*`.fontSmoothingEnabled`

This may not be supported on all available platforms. The effects of font smoothing are subtle but can significantly improve readability of the content on the screen.



Screen.height (Property)

The physical height of the screen display in pixels.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myScreen*`.height`

Property attributes:

`ReadOnly`.

Refer to:

`Screen.availHeight`

Screen.pixelDepth (Property)

The pixel Z-depth for the screen display.

Availability:	JavaScript – 1.2 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myScreen*.pixelDepth

Property attributes:

`ReadOnly`.

Refer to:

`Screen.colorDepth`

Screen.updateInterval (Property)

The screen refresh rate.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myScreen*.updateInterval

This may be important for creating animated effects. You can avoid strobing and flickering artefacts if you are aware of the refresh rate of the screen and don't try to update the display too often for the available refresh rate.

Screen.width (Property)

The physical width of the screen display in pixels.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myScreen*.wiidth

Property attributes:

`ReadOnly`.

Refer to:

`Screen.availHeight`

screenLeft (Property)

The left edge of the display screen.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myWindow`*`.screenLeft`
	IE	`screenLeft`

Refer to:

Window.screenLeft

screenTop (Property)

The top edge of the display screen.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myWindow`*`.screenTop`
	IE	`screenTop`

Refer to:

`Window.screenTop`

screenX (Property)

The X coordinate of the window within the screen display.

Availability:	JavaScript – 1.2 Netscape – 4.0
Property/method value type:	`Number primitive`

JavaScript syntax:	N	*myWindow*.screenX
	N	*myWindow*.screenX = *aCoordinate*
	N	screenX
	N	screenX = *aCoordinate*
Argument list:	*aCoordinate*	A pixel position on the screen

Refer to:

Window.screenX

screenY (Property)

The Y coordinate of the window within the screen display.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	Number primitive	
JavaScript syntax:	N	*myWindow*.screenY
	N	*myWindow*.screenY = *aCoordinate*
	N	screenY
	N	screenY = *aCoordinate*
Argument list:	*aCoordinate*	A pixel position on the screen

Refer to:

Window.screenY

Script (Definition)

A collection of source elements and optional function declarations.

Availability:	ECMAScript edition – 2

The ECMAScript standard describes a script as a collection of source elements and optional function declarations.

Any statements that are not part of a function body are considered to be part of the global code. The statements encountered as part of the global code are executed as they are parsed. Any statements that are inside a function body are simply stored and deferred for execution until the function is actually called.

See also:	Function code, function(...) ..., Global code, Script execution, Script source text, Script termination

Cross-references:

ECMA 262 edition 2 – section – 14

ECMA 262 edition 3 – section – 14

Script execution (Definition)

The process of running script.

Scripts are executed in several different ways according to the context and environment in which they are run.

Script code enclosed inside `<SCRIPT>` tags in a web page is executed as the page is rendered or when the browser generates events. Some of the code (the global code) is executed as the page is loaded. Code contained in functions is called as it is needed, however it's probably a good idea not to call function code that is defined in `<SCRIPT>` tags within the `<BODY>` from global code that is in the `<HEAD>` of a web page. Some implementations may not mind but others may fail.

Some web browsers will allow you to access objects within the document while it is being built, others will only allow access to objects once the closing `</BODY>` tag has been processed. This suggests that there is some fixing up of the internal object referencing structures that is required before the DOM tree can be traversed. An example of this is the way that MSIE handles `<DIV>` and `<OBJECT>` tags. You may not reference an `<OBJECT>` tag from a script until the `body.onLoad` event. You can sometimes access `<DIV>` blocks after they have been created but before the `</BODY>` has been reached. This behaviour may be version-1specific and also OS-specific and it may be that Macintosh and Windows versions of a browser behave differently. In the case of MSIE and Netscape Navigator, the rendering engines of both were rewritten at versions 5 and 6 respectively, and therefore many previous problems will be gone, only to be replaced by many new ones.

Scripts may be activated by a CGI interface when they are used in the middle-ware part of a server-side solution. These implementations will generally lack any kind of document model but instead will have a file system model and possible a means of reaching a database.

Scripts in embedded implementations may be triggered by a variety of situations. TV set-top boxes may trigger a JavaScript in receipt of a URL encoded into the vertical interval of the TV signal. This is a way of hiding URL data in the same way as closed caption and teletext information. Analogue TV is being phased out over the next few years but similar mechanisms will be provided with the new digital TV platforms.

Scripts used in WAP phones provided a card and stack metaphor very like HyperCard. This is embodied in WML and WScript. The scripts are the mechanism by which the pages are presented to the user. They are transmitted in a compact byte-code form, which needs to be compiled before delivery.

Warnings:

- In Navigator 2, when a window is resized, all the scripts in a page are executed again. You may not want this to happen.

See also:	`<SCRIPT EVENT="...">`, Event, Event handler in `<SCRIPT>`, Host environment, iCab, Internet Explorer, Netscape Navigator, OpenTV, Opera, PDF, Platform, Script, Script termination, Side effect, Storage duration, WebTV, WScript

Cross-references:

O'Reilly *JavaScript Definitive Guide* – page – 221-5

Wrox *Instant JavaScript* – page – 5

Script fragment (Definition)

A small portion of a script source text.

A script fragment is so small that it hardly qualifies as a script in its own right.

You might find a fragment of script in a `<SCRIPT>` block that is placed into am `<HTML>` document and executed inline as the page is loaded. It might contain only a single `document.write()` statement.

Another likely candidate for being called a script fragment is an event handler in an anchor tag. This might call a function or just set a variable value. It might be one or even several lines of JavaScript code separated by semi-colons.

See also:	Script Source Text, Statement

Cross-references:

Wrox *Instant JavaScript* – page – 16

SCRIPT object (Object/HTML)

An object that represents a `<SCRIPT>` block within the document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*mySCRIPT* = *myDocument*.all.*anElementID*
	IE	*mySCRIPT* = *myDocument*.all.tags("SCRIPT")[*anIndex*]
	IE	*mySCRIPT* = *myDocument*.all[*aName*]
	-	*mySCRIPT* = *myDocument*.getElementById(*anElementID*)
	-	*mySCRIPT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*mySCRIPT* = *myDocument*.getElementsByTagName("SCRIPT")[*anIndex*]
HTML syntax:	`<SCRIPT> ... </SCRIPT>`	

Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`charset, defer, event, htmlFor, readyState, recordNumber, src, text, type`	
Event handlers:	`onClick, onDblClick, onError, onHelp, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReadyStateChange`	

Given that you can access a `SCRIPT` object, you may be tempted to write some self-modifying code. This is not recommended. You should be able to accomplish everything you need to do in that respect with an `eval()` method and that would be more widely supported across browsers.

Accessing the script block may be useful to ascertain whether a particular function is available, although you should know that since you wrote the page yourself. On the other hand, if you imported a script block with a reference to a `.js` file, you may not know the provenance of its contents.

In the example, the source text is extracted from a script block and executed with an `eval()` function. The variable value is set according to the evaluated script which replaces the default value.

Warnings:

- This is not supported by Netscape Navigator prior to version 6.0.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT ID="ONE">
block = "ONE";
</SCRIPT>
<SCRIPT ID="TWO">
block = "TWO";
</SCRIPT>
<SCRIPT ID="THREE">
block = "THREE";
</SCRIPT>
<SCRIPT ID="FOUR">
block = "FOUR";
mySourceText = eval(document.scripts.THREE.text);

document.write(block);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Document.scripts[]`,`Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
charset	1.5 +	-	6.0 +	-	-	1 +	-	Warning
defer	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
event	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly
htmlFor	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly
readyState	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
recordNumber	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly
src	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly
text	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly
type	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5+	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onError	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onLoad	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onReadyStateChange	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

SCRIPT.charset (Property)

The character set that a script conforms to.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	N	mySCRIPT.charset

This would contain the character set being used by the script file referred to by the SRC="..." HTML tag attribute. For example the value "iso-8859-1" is likely to be returned but the local variant of the browser and OS may affect the value you get.

This property might contain a value such as:

```
csISO5427Cyrillic
```

Details of other aliases can be located at the IANA registry (see web reference below).

Warnings:

- This is not supported by Netscape Navigator prior to version 6.0.

Web-references:

ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

SCRIPT.defer (Property)

Whether the execution of a script object is to be deferred during page loading to speed the construction of a page.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	-	*mySCRIPT*.defer

Deferring script execution can sometimes get round some tricky problems with accessing a page before it is ready. There are certain stages that the page object model undergoes which are not apparent from the outside.

For example, when loading a page a complex object hierarchy needs to be built. That is not going to be complete until the closing </BODY> tag is encountered. Before that time, certain objects may be accessed but others may not. For instance, prior to the body closure, you cannot use a document.write() to insert an ActiveX <OBJECT> tag using inline coding techniques. You can do a <DIV> block replacement when the body has closed but an error results if you try to do this too soon.

Warnings:

- This is not supported by Netscape Navigator prior to version 6.0.

See also:	XML.defer

SCRIPT.event (Property)

Scripts can be associated with events in the event model.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*mySCRIPT*.event

Although this is not portable, it might be very useful as a debugging aid when developing scripts.

Warnings:

- This is not supported by Netscape Navigator prior to version 6.0.

Property attributes:

`ReadOnly`.

SCRIPT.htmlFor (Property)

The element `ID` associated with the `FOR="..."` HTML tag attribute in the `<SCRIPT>` tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*mySCRIPT*.htmlFor

Scripts can be bound to a named document element object. This returns the `ID` value for the HTML element to which the script object is bound.

Warnings:

- This is not supported by Netscape Navigator prior to version 6.0.

Property attributes:

`ReadOnly`.

SCRIPT.readyState (Property)

The current status disposition of a <SCRIPT> block as it is being loaded.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*mySCRIPT*.readyState

This property reflects the loading state of a <SCRIPT> block that is loaded from a source file.

Sometimes, you can design scripts to execute while the document is downloading, inline scripts for example. At that time, you may even be able to trigger interval timed deferred executions as well.

If it is important that the document has completed loading before execution, you can check this property for one of the following values:

State	Value
uninitialized	The object is first instantiated but has not begun loading.
loading	The object has commenced loading.
loaded	The object has completed loading.
interactive	The object is loaded but not yet closed but is ready to handle interaction.
complete	The object body has been closed and the loading is finished.

An object may not need to reflect the complete status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an OBJECT object that represents an <OBJECT> tag until the <BODY> has completed loading. This is because the ActiveX object construction requires a complete document body structure to attach itself to.

Every time this readyState value changes, it triggers an onReadyStateChange event call-back.

See also:	onReadyStateChange

Property attributes:

ReadOnly.

SCRIPT.recordNumber (Property)

The record number within the data set that created the script's content.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Property/method value type:	Number primitive	
JavaScript syntax:	IE	*mySCRIPT*.recordNumber

This is a property that is part of the MSIE data binding support. It contains an integer value that is the record number within the data set that created this object.

This is useful when you are building pages with ASP and Active Data Objects (ADO).

Warnings:

- This is not supported by Netscape Navigator.

See also:	Active Server Pages, ADO

Property attributes:

ReadOnly.

SCRIPT.src (Property)

The URL where the source text for a script block is kept.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*mySCRIPT*.src

This property is read-only. There would be little point in reloading the script code from a new URL and it could be quite dangerous to allow this to happen under script control.

MSIE may yet still provide some access to the contents of a script object in a way that lets you change the script source contained in a <SCRIPT> block. It certainly lets you read the source text.

Warnings:

- This is not supported by Netscape Navigator.

See also:	<SCRIPT SRC="...">

Property attributes:

ReadOnly.

SCRIPT.text (Property)

The textual content of a script block.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*mySCRIPT*.text

You can use this to check for the existence of a function before attempting to call it.

Warnings:

- This is not supported by Netscape Navigator prior to version 6.0.

Property attributes:

`ReadOnly`.

SCRIPT.type (Property)

The MIME type of the `<SCRIPT>` block's content.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*mySCRIPT*.type
HTML syntax:	`<SCRIPT TYPE="...">`	

The MIME type of the script object is accessible through the value of this property.

Refer to the MIME type topic for details of the available MIME types you will likely see in this property.

Warnings:

- This is not supported by Netscape Navigator prior to version 6.0.

See also:	`<SCRIPT TYPE="...">`, `<STYLE TYPE="...">`, `StyleSheet.type`

Property attributes:

`ReadOnly`.

Script Source Text (Definition)

The original source text that is interpreted and executed.

Availability:	ECMAScript edition – 2

JavaScript implementations that conform to ECMAScript are embodied as a source text that a human can read and edit in a programming environment or text editor. The text of the script source must be expressed using the lower 128 character entities in the Unicode version 2.0 character set.

You can use other Unicode characters but only within comments and string literals.

Any Unicode character can be represented with an escape sequence composed only of characters in the lower 128 range. The escape sequence follows the normal tradition of specifying the character value (in hex) using its numeric position within the character set, like this:

```
\u47AD
```

Within a comment, such an escaped Unicode character is effectively ignored while within a string literal, it contributes a single character to the string.

Although all the characters in a conforming script are Unicode, they are treated with any context dependent interpretation as specified in the Unicode standard. The value of a 16 bit character is sometimes called a code point.

In JavaScript, escape sequences inside comments are never interpreted. That's actually a good thing, because the `\u000A` escape sequence is a line terminator and if you had one and commented out the line suddenly your script would break.

See also:	Comment, Escape sequence (\), Formal Parameter List, `function( ... ) ...`, Lexical convention, Script, Script fragment, Variable Declaration, Variable instantiation

Cross-references:

ECMA 262 edition 2 – section – 6

ECMA 262 edition 3 – section – 6

<SCRIPT SRC="..."> (HTML Tag Attribute)

The URL to access an insertable fragment of JavaScript contained in an include file.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
HTML syntax:	`<SCRIPT SRC="..."></SCRIPT>`	
Argument list:	`...`	A URL to reach an includable `.js` file.

This provides a way to include JavaScript from an external file and share some common functionality among several pages. No data is transported from page to page apart from that which is in the included file as assignment statements, so this is not a means of maintaining state between pages.

Although this behaves in the same way as script that is in the HTML page itself, some people prefer not to call this inline code because it's held in a separate file. It depends whether you are referring to the location of the code when you use the term inline or whether you are talking about how it is executed; included script code is executed inline.

You can refer to javascript `.js` files on your local client-side hard disk as long as the page is not being requested from a web server. To access a local client file in a page coming from a web server would be to break the security regimes established to prevent client systems being hacked by intruders.

This is a really useful technique for debugging because you can build an entire library of object disassembly and diagnostic tools and include them in a script block which you can then eliminate when the code goes into production use.

A lot of the undocumented features of the browsers were uncovered in this way.

Warnings:

- Included files sometimes do not work as expected. Browsers do not always support the capability, or may not be configured to accept the `application/x-JavaScript` MIME type as an executable file with a `.js` file extension.
- Note also that although you can examine the properties of `SCRIPT` objects within the MSIE browser, the `SCRIPT` objects that are created as a result of a `SRC="..."` include will not enumerate their properties without causing a run-time error in your script.

Example code:

```
<SCRIPT SRC="include.js">
</SCRIPT>
-----------------------------------------------------
// This content goes into include.js
function getBrowserType()
{
   var myUserAgent;
   var myMajor;
   myUserAgent   = navigator.userAgent.toLowerCase();
   myMajor       = parseInt(navigator.appVersion);
```

```
    if( (myUserAgent.indexOf('mozilla')    != -1) &&
        (myUserAgent.indexOf('spoofer')    == -1) &&
        (myUserAgent.indexOf('compatible') == -1) &&
        (myUserAgent.indexOf('opera')      == -1) &&
        (myUserAgent.indexOf('webtv')      == -1)
       )
    {
       return "N";
    }
    if (myUserAgent.indexOf("msie") != -1)
    {
       return "msie";
    }
    return "other";
}

document.write(getBrowserType());
```

See also:	`.js`, `<SCRIPT ARCHIVE="...">`, `<SCRIPT>`, Adding JavaScript to HTML, File extensions, Inline script, MIME types, `SCRIPT.src`, Security policy

Cross-references:

Wrox *Instant JavaScript* – page 42

Script termination (Definition)

The act of halting the execution of a script.

Scripts may terminate in an expected and normal way or abnormally due to some problem. The predominant reason for premature aborts of scripts is some kind of programmer error, most likely a syntax error or a reference to a non-existent object.

Web browsers behave in different ways and you may be able to add development tools around the browser to catch the exception as it happens and carry out some debugging.

Generally speaking the debugging tools for JavaScript are weak by comparison with those that you use with a compiled language. Some environments provide better tools than others do.

See also:	Environment, Error handling, Execution environment, Script, Script execution, Storage duration

<SCRIPT TYPE="..."> (HTML Tag Attribute)

The MIME type for a block of script code.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
HTML syntax:	`<SCRIPT TYPE="...">` *`someCode`* `</SCRIPT>`	
Argument list:	`...`	A MIME type that signifies JavaScript source text.
	`someCode`	Some script source text

This is an alternative method of selecting the interpreter to be used for the <SCRIPT> block. It has limited support prior to the version 4.0 browsers and so it may be less portable for a while.

Warnings:

- If a browser does not understand this attribute and if there is not a corresponding LANGUAGE attribute that it does understand, it is possible that the script block will be ignored and the script code may not be executed.

Example code:

```
<SCRIPT TYPE="text/JavaScript">document.write("Basic functionality")</SCRIPT>
```

See also:	<SCRIPT LANGUAGE="...">, <SCRIPT>, <STYLE TYPE="...">, MIME types, SCRIPT.type, StyleSheet.type, text/JavaScript

Cross-references:

Wrox *Instant JavaScript* – page 42

</SCRIPT> (Pitfall)

Problems with closing <SCRIPT> tags.

You cannot use the string '</SCRIPT>' within an inline JavaScript fragment. Even if it is enclosed inside quotation marks, it will still be seen by the parser and interpreted as a closure to the <SCRIPT> tag. You will need to hide it by constructing the string from component parts and using concatenation techniques to manufacture the string you need.

If you need to say this:

```
var myScriptTag = '</SCRIPT>';
```

Then you should do this:

```
var myScriptTag = '<' + '/SCRIPT' + '>';
```
which should hide the tag from the parser.

Another alternative is to escape the slash character with a backslash like this:

```
var myScriptTag = '<\/SCRIPT>';
```

Look at that previous line carefully, and see how the forward slash is preceeded by a backslash. After a long day cranking code out that might look like an upper case V, so it might be best to use the concatenation technique.

See also:	<SCRIPT>, Pitfalls

Cross-references:

Wrox *Instant JavaScript* – page 45

<SCRIPT ARCHIVE="..."> (HTML Tag Attribute)

The URL to access an archive containing insertable fragments of JavaScript contained in a single file.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0	
HTML syntax:	`<SCRIPT ARCHIVE="..." SRC="..."></SCRIPT>`	
Argument list:	`...`	A URL to reach an includable `.js` or `.jar` file

This tag attribute allows an archive file to be specified. This allows a whole collection of `.js` files to be shipped as a single unit. The required `.js` file can then be included by extracting it from the archive. If the archive is requested from the web server, it will likely persist in the cache and therefore some time is saved by collecting these archives and referring to items contained within them.

The `ARCHIVE` tag attribute has no use on its own in this context and you must use it with the `SRC` attribute.

Archives are called `.jar` files and are basically a zip-compressed collection of `.js` files. You will probably find one of the Java development environments contains all the tools you need to create these archives in a straightforward way. Actually you don't need anything more complex than a text editor and a zip compression tool.

You can do this manually if you follow these steps:

- Create your collection of `.js` files
- Collect them into a zipped archive file whose file extension is `.jar`
- Construct some HTML to include them
- Call the functions as needed

Refer to the example for script and HTML code. Create the two files `test1.js` and `test2.js` and store them in an archive called `test.jar`. Then create `test.html` with its `<SCRIPT>` tags to call in the files from the archive. When you run it, you can see both functions are called during the `document.write()` methods.

Example code:

```
// Save this into file test1.js
function test1()
{
   return "Test 1";
}
------------------------------------------------------------
// Save this into file test2.js
function test2()
{
   return "Test 2";
}
```

```
------------------------------------------------------------
<!-- Save this as file test.html -->
<HTML>
<HEAD>
<SCRIPT ARCHIVE="./test.jar" SRC="test1.js"></SCRIPT>
<SCRIPT ARCHIVE="./test.jar" SRC="test2.js"></SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
document.write(test1());
document.write("<BR>");
document.write(test2());
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	.jar, .js, <SCRIPT SRC="...">, <SCRIPT>, Adding JavaScript to HTML, File extensions, Inline script, MIME types, Security policy

<SCRIPT EVENT="..."> (HTML Tag Attribute)

A tag attribute to associate a script block with an event to be handled.

Availability:	JScript – 3.0 Internet Explorer – 4.0

This HTML tag attribute is quite useful when using ActiveX controls in web pages. You can use this to attach a fragment of script to an event so that the screen gets updated.

Here is a skeleton of some HTML that attaches a script to an object that has been embedded:

```
<SCRIPT FOR="Xbutton" EVENT="Click()">
// Do some kind of stuff in here as a
// result of the ActiveX calling this
<SCRIPT>
<OBJECT ID="Xbutton" CLASSID="..." CODEBASE="..." STYLE="...">
<PARAM NAME="..." VALUE="...">
</OBJECT>
```

The CLASSID, CODEBASE, and other parameters depend on the ActiveX control you are embedding. The point to make here is that as the page is loaded, the control will be displayed and when the user clicks on it, the browser makes the association by mapping the FOR="..." HTML tag attribute in the <SCRIPT> tag to the ID="..." attribute of the <OBJECT> tag. Then the event that the control triggers is mapped to the EVENT="..." HTML tag attribute of the <SCRIPT> tag.

You can create a whole set of <SCRIPT> blocks, one for each event and control you expect to use. This means the browser does the mapping and dispatching of events for you.

See also:	<SCRIPT>, Event handler in <SCRIPT>, Script execution

<SCRIPT FOR="..."> (HTML Tag Attribute)

A tag attribute to associate a script block with an input element or for mapping script blocks to objects embedded in web pages.

Availability:	JScript – 3.0 Internet Explorer – 4.0
See also:	`<SCRIPT EVENT="...">`, `<SCRIPT>`, Script execution

Refer to:

Event handler in `<SCRIPT>`

<SCRIPT ID="..."> (HTML Tag Attribute)

Script blocks can be given `ID` values so they can be identified within the document scripts array.

If you can identify a script block by its `ID` value, you should be able to locate the object and at least read the contents of the script block. Some browsers may let you change the script block but it has always been recommended practice to avoid self-modifying code.

See also:	`Document.scripts[]`

<SCRIPT LANGUAGE="..."> (HTML Tag Attribute)

The required version of JavaScript to interpret the enclosed code.

HTML syntax:	`<SCRIPT LANGUAGE="...">` *`someCode`* `</SCRIPT>`	
Argument list:	`...`	The script language to use for this block of script source
	`someCode`	Some script source text

As you embed the script code into an HTML page with the `<SCRIPT>` tag, you can indicate by means of the `LANGUAGE` attribute which version of JavaScript (or indeed other scripting languages) the interpreter should use to process the script. This is subtle and allows various aspects of the language to be switched so that they behave differently according to the version selection.

It also provides a way to hide JavaScript written according to newer syntax conventions from older browsers that cannot cope with it. In general, you should always try to specify the lowest version of JavaScript to achieve maximum portability.

JavaScript version 1.2 implemented some different capabilities regarding equality tests where the operands were different types. Selecting `LANGUAGE="JavaScript"` as opposed to `LANGUAGE="JavaScript1.2"` affects how these tests are carried out when the script is executed.

The following values are legal for the <SCRIPT> tag's LANGUAGE attribute:

Attribute Value	Description
Nothing, attribute omitted	Basic JavaScript functionality.
JavaScript	Basic JavaScript functionality.
JavaScript1.1	Version 1.1 language capabilities.
JavaScript1.2	Version 1.2 language capabilities.
JavaScript1.3	Version 1.3 language capabilities.
JavaScript1.4	Version 1.4 language capabilities.
JavaScript1.5	Version 1.5 language capabilities.
VBScript	Visual BASIC scripting in MSIE browsers.
Tcl	In the HTML 4.0 specification, Tcl is used as an example.

The example below will display the text 1.3 in a Netscape 4.7 browser and the value 1.4 in version 5 of MSIE for Macintosh.

Note with this technique that you should ensure you test for a high enough version. The browsers will execute the versions indicated. If you only test up to version 1.2, then the variable assignment is never going to reflect a 1.4 version capability.

Warnings:

- If a browser does not support the specified language, it may not execute the script block, even with a degraded version of the interpreter.
- Be aware that Netscape 4 supports some special capabilities in JavaScript version 1.2 mode that are not strictly correct according to the ECMA standard nor are they compatible with earlier versions of Netscape Navigator and other browsers. If you find that you need to turn on JavaScript version 1.2 with the LANGUAGE attribute, check your scripts for portability very carefully.

Example code:

```
<!-- JavaScript version detector --->
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">    myVersion = "Generic";</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.0"> myVersion = "1.0";</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1"> myVersion = "1.1";</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2"> myVersion = "1.2";</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.3"> myVersion = "1.3";</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.4"> myVersion = "1.4";</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.5"> myVersion = "1.5";</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
document.write(myVersion);
</SCRIPT>
</BODY>
</HTML>
```

See also: <META>, <SCRIPT TYPE="...">, <SCRIPT>, Compatibility, Element.language

Cross-references:

Wrox *Instant JavaScript* – page 42

<SCRIPT> (HTML Tag)

A container for JavaScript in an HTML page.

HTML syntax:	`<SCRIPT> someCode </SCRIPT>`	
Argument list:	*someCode*	Some script source text

The <SCRIPT> tag is how the JavaScript code is embedded into a web page. There are several ways to do this. Note that the <SCRIPT> tag must have an associated closing </SCRIPT> tag at the end of the script source text. The following attributes may be useful:

- LANGUAGE
- SRC
- ARCHIVE
- TYPE

You can place the <SCRIPT> tag in the <HEAD> or <BODY> section of the page.

In the <HEAD> context, it is expected to provide some support to the rest of the page so you might place useful functions and event handlers here. You can also put some global code here and it will get executed inline, probably initializing some global variables. The intent is for the <HEAD> block to contain only meta-information about the document. This implies you should not place a `document.write()` method into the <HEAD> area such that it will be executed during page loading. However, both the Netscape and MSIE browsers will allow the `document.write()` if it is called during page loading even if it is located in the <HEAD> block.

It makes some sense to be sure that if you do a `document.write()` in the <HEAD> block, that you make sure it writes something sensible. For example, you can write the document title from a script that is executed inline. If you just write some textual output, the browsers are at least smart enough to place it into the page body.

Because JavaScript is interpreted, it needs to have any functions declared before they are called. However, this means they must be declared chronologically before they are called. This is not the same as positionally defining them before they are called because the page content may be traversed several times. For example, an event handler can probably be placed anywhere because very few events happen before the </BODY> closure happens. Nevertheless, it is still probably good practice to locate any functions that you can in <SCRIPT> blocks placed in the <HEAD> area.

You can place <SCRIPT> tags throughout the <BODY> of the document. These might also contain functions but are more likely to contain inline code to be executed as the page is loading. Although they are in different <SCRIPT> tag blocks, they are all conceptually part of the same script.

If your inline script code is going to do any `document.write()` calls to modify the HTML as the page is loaded, then this is the optimum place to put the code. It's quite sensible to break the code into smaller <SCRIPT> blocks and place them appropriately throughout the document. If the code starts to become complicated, then factor some of it into functions placed in the <HEAD> area and then call it as needed from the <BODY> area.

If you are using a frame-set, you can put the <SCRIPT> block after the <HEAD> tag but before the <FRAMESET> tag. You could write the entire <FRAMESET> description at this point using document.write() methods. You could do that with the entire <BODY> content too.

You can break your script code into smaller blocks, each one associated with a different <SCRIPT></SCRIPT> area and, if necessary, each can be executed in a different version of JavaScript. They will each have a different execution context and the scope chain may be affected, although global variables should be reachable from anywhere.

When the browser encounters a <SCRIPT> tag, it pauses the processing of the HTML page description and executes the <SCRIPT> tag's source code. That may affect subsequent HTML output anyway, and may generate some HTML to be placed into the page at the point where the <SCRIPT> block appears. Any lengthy script evaluation is going to slow down the display of your page. You should defer any lengthy processing until you can use some sleight of hand to hide it. For example, perhaps you can wait until the <BODY> tag is closed and then activate some processing with a <BODY ONLOAD="...">handler. This may be an issue if you are using included .js files since they will need to be requested and fetched from a web server.

You can build event handlers and associate them with the event by means of the <SCRIPT> tag attributes, but this only works in MSIE.

Warnings:

- Be aware that if you place <SCRIPT> blocks inside the <HEAD> of a document, you may be able to initialize some data structures but you certainly won't be able to access any objects that belong to the <BODY> since they won't yet exist.
- Note that during page loading, until you have reached the closing </BODY> tag, the page may be in some intermediate state where objects and memory locations are not locked down. This may cause some difficulties in writing to the document or changing the content of <DIV> blocks. In particular, you cannot inline document.write() into the content of an <OBJECT> block. Until the page is completed, the <OBJECT> is not properly linked into a structure in which you can access it from JavaScript, this is the case with MSIE version 4 browsers, at least.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myResult = 100;
myExpr = myResult %= 1000;
document.write(myResult);
document.write("<BR>");
document.write(myExpr);
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	</SCRIPT>, <META>, <NOSCRIPT>, <SCRIPT ARCHIVE="...">, <SCRIPT EVENT="...">, <SCRIPT LANGUAGE="...">, <SCRIPT SRC="...">, <SCRIPT TYPE="...">, <STYLE TYPE="...">, Adding JavaScript to HTML, Document.scripts[], Host environment, HTML file, String, Web browser

Cross-references:

Wrox *Instant JavaScript* – page 42

ScriptArray object (Object/browser)

A collection of script blocks belonging to a document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myScriptArray* = *myDocument*.scripts
Object properties:	length	
Object methods:	item()	

In the example, the document contains several script blocks. The script source is extracted and formatted with an escape() function and line breaks are reinserted by means of the String.split() and Array.join() methods.

Warnings:

- This is not supported by Netscape Navigator.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<TABLE BORDER=1>
<TH>Index</TH>
<TH>ID</TH>
<TH>Text</TH>
<SCRIPT ID="ONE">
block1 = "ONE";
</SCRIPT>
<SCRIPT ID="TWO">
block2 = "TWO";
</SCRIPT>
<SCRIPT ID="THREE">block3 = "THREE";</SCRIPT>
<SCRIPT ID="FOUR">
block4 = "FOUR";
```

```
myLength = document.scripts.length;
for (myEnumerator=0; myEnumerator<myLength; myEnumerator++ )
{
   mySourceText = escape(document.scripts[myEnumerator].text);
myArray = mySourceText.split("%0D");

   document.write("<TR><TD>");
   document.write(myEnumerator);
   document.write("</TD><TD>");
   document.write(document.scripts[myEnumerator].id);
   document.write("</TD><TD>");
   document.write(myArray.join("%0D<BR>"));
   document.write("</TD></TR>");
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	Collection object, `Document.scripts[]`, `ScriptArray.length`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
length	-	3.0 +	-	4.0 +	-	-	Warning, ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
item()	-	3.0 +	-	4.0 +	-	-	-

ScriptArray.item() (Method)

An item selector for accessing a single script within the collection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Script object`	
JavaScript syntax:	IE	*myScriptArray*`.item(anIndex)`
	IE	*myScriptArray*`.item(`*aSelector*`)`
	IE	*myScriptArray*`.item(`*aSelector*`, `*anIndex*`)`
Argument list:	*anIndex*	A zero based index into the collection
	aSelector	A textual value that selects all matching objects

Refer to:

`Collection.Item()`

ScriptArray.length (Property)

The number of script blocks in the current document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myDocument*`.scripts.length`

Warnings:

- This is not supported by Netscape Navigator.

See also:	`Collection.length`, `ScriptArray` object

Property attributes:

`ReadOnly`.

ScriptEase (Product)

A standalone JavaScript interpreter sold by Nombas Inc.

See also:	Server-side JavaScript

Refer to:

Standalone JavaScript

ScriptEngine() (Function)

A special MSIE globally available function that describes the scripting engine currently installed for use with the browser.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`ScriptEngine()`

The following values will be returned by this function depending on the context in which it is called:

- `JScript`
- `VBA`
- `VBScript`

See also:	`Navigator.appMinorVersion`

ScriptEngineBuildVersion() (Function/global)

A special MSIE globally available function that describes the build version of scripting engine currently installed for use with the browser.

Availability:	JScript – 2.0 Internet Explorer – 4.0
Property/method value type:	`String primitive`

This is only available in the MSIE browser. It is useful if you are developing scripts and it is possible that you could build a reference to this into a form that is submitted as part of an error handler. That way you might determine what the error is, what caused it and note the build number of the script interpreter. This may then yield a pattern. An essential part of the fault diagnosis process involves the search for a pattern in the failures of a system. It is very possible that a certain build of the interpreter could manifest a bug which is not present in other builds.

See also:	`Navigator.appMinorVersion`

ScriptEngineMajorVersion() (Function/global)

A special MSIE globally available function that describes the major version number of the scripting engine currently installed for use with the browser.

Availability:	JScript – 2.0 Internet Explorer – 4.0
Property/method value type:	`String primitive`

See also:	`Navigator.appMinorVersion`

Refer to:

`ScriptEngineBuildVersion()`

ScriptEngineMinorVersion() (Function/global)

A special MSIE globally available function that describes the minor version number of the scripting engine currently installed for use with the browser.

Availability:	JScript – 2.0 Internet Explorer – 4.0
Property/method value type:	`String primitive`

See also:	`Navigator.appMinorVersion`

Refer to:

`ScriptEngineBuildVersion()`

Scriptlet (Definition)

A scripting component within the Windows Script Host environment.

Scriptlets are built in the WSH environment or in the web browser. They have evolved over time and become useful for a variety of purposes.

Those items that used to be called scriptlets in the MSIE browser have evolved into what is now called HTML components or HTCs for short.

The original kind of scriptlet (nowadays called a version 1 scriptlet) should now be referred to as a DHTML scriptlet. It is used to construct an HTML object with XML and JavaScript.

A version 2 scriptlet is compiled from its XML and JavaScript source into an ActiveX object which can be used in contexts other than web pages.

Standalone JavaScript scripts that run in WSH are not scriptlets but are simply scripts.

Scriptlets can be built in a variety of ways. Although we are mainly concerned with JavaScript, you can use Perl or VBScript, too. They all use a similar XML framework.

See also:	`.htc`, HTML Component

Cross-references:

Wrox *Instant JavaScript* – page – 284

Wrox *Professional JavaScript* – page – 776

Web-references:

news://msnews.microsoft.com/public.scripting.scriptlets
http://msdn.microsoft.com/scripting;
http://msdn.microsoft.com/workshop/languages/clinic/xmlscript.asp
http://wsh.glazier.co.nz/

scroll() (Method)

An alias for the `window.scroll()` method.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0
Deprecated:	Yes
Property/method value type:	`undefined`

JavaScript syntax:	-	*myWindow*.scroll(*aPositionX*, *aPositionY*)
	-	scroll(*aPositionX*, *aPositionY*)
Argument list:	*aPositionX*	A position in pixels
	aPositionY	A position in pixels

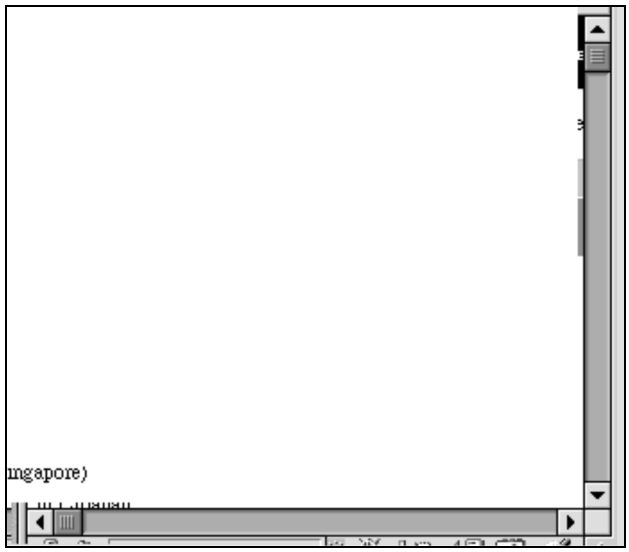
Refer to:

Window.scroll()

scrollbars (Property)

An alias for the window.scrollbars property.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	Bar object	
JavaScript syntax:	-	*myWindow*.scrollbars
	-	scrollbars



See also:	Bar object

Property attributes:

ReadOnly.

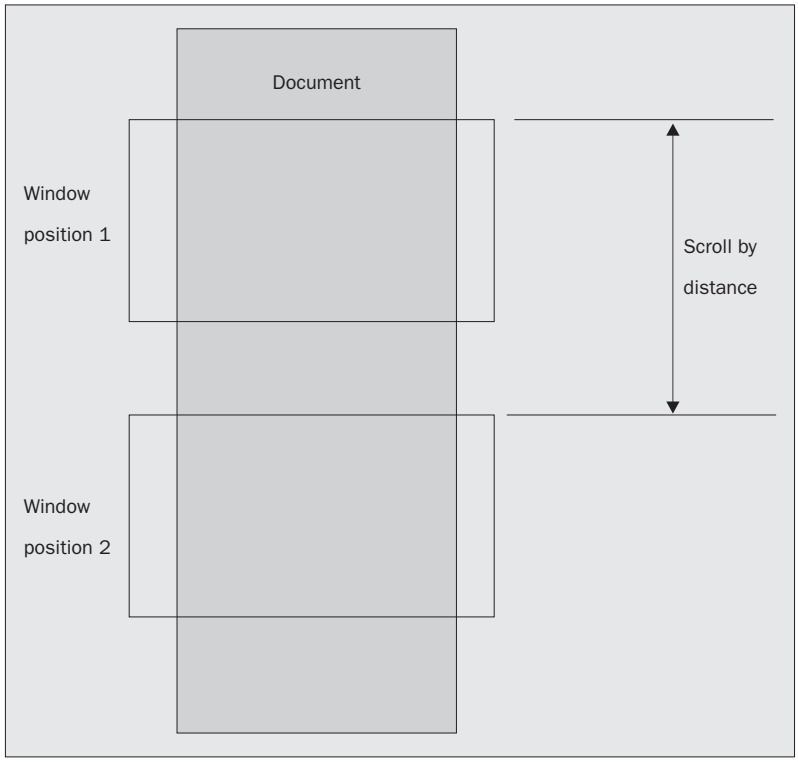
Refer to:

Window.scrollbars

scrollBy() (Method)

An alias for the `window.scrollBy()` method.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*`.scrollBy(`*anOffsetX*`,` *anOffsetY*`)`
	-	`scrollBy(`*anOffsetX*`,` *anOffsetY*`)`
Argument list:	*anOffsetX*	A distance in pixels
	anOffsetY	A distance in pixels



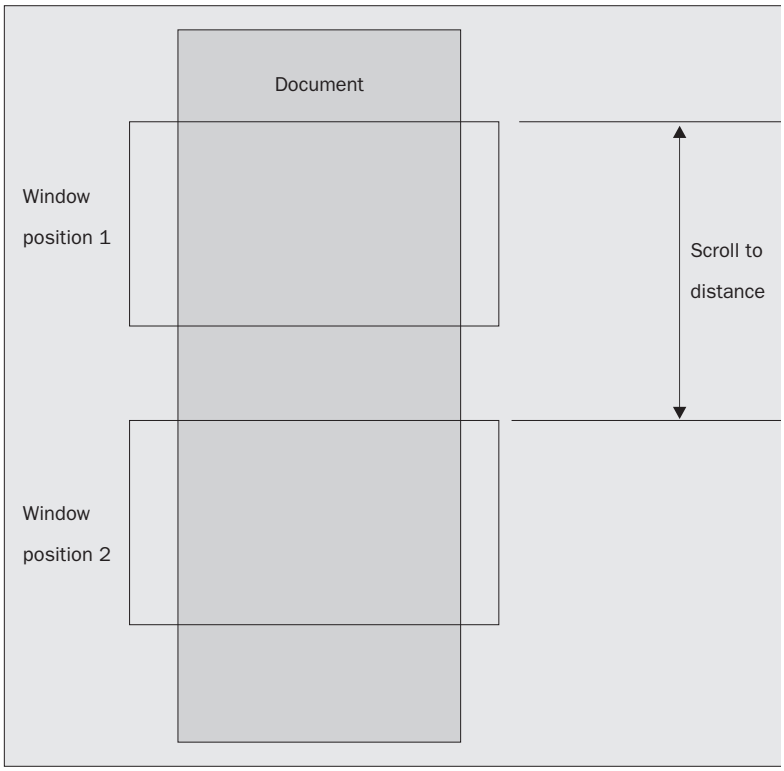
Refer to:

`Window.scrollBy()`

scrollTo() (Method)

An alias for the `window.scrollTo()` method.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*`.scrollTo(`*aPositionX*`,` *aPositionY*`)`
	-	`scrollTo(`*aPositionX*`,` *aPositionY*`)`
Argument list:	*aPositionX*	A location in pixels
	aPositionY	A location in pixels



Refer to:

`Window.scrollTo()`

secure (Property)

A flag indicating that a window was loaded from a secure source.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myWindow*`.secure`
	N	`secure`
Argument list:	*false*	Window is not currently securely served
	true	Window was loaded from a secure location

Property attributes:

`ReadOnly`.

Refer to:

`Window.secure`

Security policy (Definition)

The rules about what can access local client-side resources.

The basic approach to security in JavaScript is to disallow anything that might be risky, which is quite straightforward in the context of a web browser. However, there are ways to work around this with signed scripts and calls to Java applets and ActiveX controls which may or may not be secure and in any case are platform dependent.

JavaScript is now beginning to be used outside the browser and so scripts may be executed in all kinds of contexts:

- Server side
- Unix command line shell
- Desktop standalone
- TV set-top box
- PDF forms
- WAP phones

Each of these has its own needs and limitations regarding security so there is not one overall approach that works everywhere but rather a variety of techniques that apply on a platform by platform basis. This means that particular fragments of script may work in one context but not in another.

In the browser, all networking capabilities are disabled. This is not so with some of the stand-alone or some of the command-line shell environments where JavaScript may run.

In Netscape 2, the security was very lax allowing JavaScript to send mail purporting to be from another user.

Netscape 3 improved security while version 4 is built around a complete new security model.

MSIE supports different security models, in particular the Authenticode model that allows for signed ActiveX objects to be used.

See also:	`.jar`, `<SCRIPT ARCHIVE="...">`, `<SCRIPT SRC="...">`, AuthentiCode, Code signing, Data-tainting, `Document.domain`, `Document.links[]`, `Global` object, https: URL, Netscape Enterprise Server, Restricted access, Same origin, snews: URL, telnet: URL, `UniversalBrowserAccess`, `UniversalBrowserRead`, `UniversalBrowserWrite`, `UniversalFileRead`, `UniversalPreferencesRead`, `UniversalPreferencesWrite`, `UniversalSendMail`

Select object (Object/HTML)

A drop-down menu containing a list of `<OPTION>` items. These are used in forms to build menus and pop-ups. They may select single items or multiple items.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`mySelect = myDocument.all.anElementID`
	IE	`mySelect = myDocument.all.anElementID.elements[anIndex]`
	IE	`mySelect = myDocument.all.tags("SELECT")[anIndex]`
	IE	`mySelect = myDocument.all[aName]`
	-	`mySelect = myDocument.getElementById(anElementID)`
	-	`mySelect = myDocument.getElementsByName(aName)[anIndex]`
	-	`mySelect = myDocument.getElementsByTagName("SELECT")[anIndex]`
	-	`mySelect = myForm.aSelectName`
	-	`mySelect = myForm.elements[anIndex]`
	-	`mySelect = myForm[anIndex]`
HTML syntax:	`<SELECT> ... </SELECT>`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The name attribute of an element
	anElementID	The `ID` attribute of an element
	aFormIndex	A reference to a particular form in the forms collection

Object properties:	accessKey, dataFld, dataSrc, form, length, multiple, selectedIndex, size, tabIndex, type, value
Object methods:	add(), remove(), tags()
Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onChange, onDragStart, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onSelectStart
Collections:	options[]

Many properties, methods and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "`Input`" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

Unusually, there actually is a `Select` object class where most other kinds of input are instances of an `Input` object.

Event handling support via properties containing function objects was added to `Select` popup objects at version 1.1 of JavaScript.

Unlike MSIE, the Netscape Navigator implementation of this object type does not support the `click()` method.

The MSIE instance of this object is actually a `SELECT` object and not a `Select` object. This is another example of class naming differences between browsers that may cause problems later.

Warnings:

- Note that this `FormElement` object type does not have a `value` property. You may need to make allowances for that in generic form object handlers. Its value is reflected by the option item that is currently selected. This `FormElement` object is not a sub-class of the `Input` object as many other `FormElements` are.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="RESULT">???</DIV>
<FORM>
<HR>
<SELECT ID="IN1">
<OPTION VALUE="0">Sunday
<OPTION VALUE="1">Monday
<OPTION VALUE="2">Tuesday
<OPTION VALUE="3">Wednesday
<OPTION VALUE="4">Thursday
<OPTION VALUE="5">Friday
<OPTION VALUE="6">Saturday
</SELECT>
<INPUT TYPE="button" VALUE="CLICK ME" onClick="clickMe()">
```

```
</FORM>
<SCRIPT>
//MSIE Only
function clickMe()
{
   selectedValue  = document.all.IN1.value;
   document.all.RESULT.innerText = selectedValue;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Element` object, `Form.elements[]`, `FormElement` object, `Input` object, `Input.accessKey`, `onChange`, `Option` object, `OptionsArray` object, `response.getOptionValue()`, `response.getOptionValueCount()`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`accessKey`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`dataFld`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`dataSrc`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`form`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`length`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	ReadOnly.
`multiple`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`selectedIndex`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`size`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`tabIndex`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly.
`value`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
add()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
remove()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
tags()	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onChange`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Select.add() (Method)

Adds a new option object to a select list collection.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*mySelect*.add(*anElement*, *anIndex*)
Argument list:	*anElement*	The option element to add
	anIndex	The index to insert it at

Select.length (Property)

The length of a select block for a popup menu.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*mySelect*.length

Property attributes:

`ReadOnly`.

Refer to:

`Collection.length`

Select.multiple (Property)

A flag indicating whether a select block can have multiple or single items only selected.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*mySelect*`.multiple`
HTML syntax:	`<SELECT MULTIPLE>`	

If this flag is set to the Boolean `true` value, then several items in the list of options can be selected simultaneously. They will be passed back in the form data as a comma separated list when the form is sumitted.

See also:	`Select.options[]`, `Select.selectedIndex`, `Select.size`

Select.options[] (Collection)

An array of options objects one each per menu item.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`Collection` object	
JavaScript syntax:	-	*mySelect*`.options`

You can create new option items dynamically with the `Option()` constructor. They can then be assigned into the array of options. If you want to remove one, then simply assign `null` to its array entry and the option will be removed from the popup menu.

See also:	`Option()`, `Option.index`, `OptionsArray` object, `Select.multiple`, `Select.selectedIndex`

Property attributes:

`ReadOnly`.

Select.remove() (Method)

Remove an item from a select list collection.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*mySelect*.remove(*anIndex*)
Argument list:	*anIndex*	The option item to remove

Select.selectedIndex (Property)

The index of the selected item in a <SELECT> block.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*mySelect*.selectedIndex

This is the index number of the Option object in the options[] collection belonging to the receiving Select object. Note that this index number is only good while the options collection is not modified. Adding new options or deleting options will change the ordering of the collection.

Feedback from several people suggests that in the case of a multiple selection taking place this property returns the index of the first selected item.

See also:	Select.multiple, Select.options[]

Select.size (Property)

The number of items currently chosen in the select popup.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*mySelect*.size

This is an integer value which is likely to be `1` most of the time. It is only meaningful if the `Select.multiple` property is `true` and more than one item has been selected.

Here is a workaround for other browsers that don't have this property, kindly donated by Jon Stephens:

```
<SCRIPT>var mySize=0;for(var i=0; i<mySelect.options.length; i++){   mySize +=
mySelect.options[i].selected ? 1 : 0;}mySelect.size = mySize;</SCRIPT>
```

See also:	`Select.multiple`

Select.tags() (Method)

A method for retrieving collections of objects belonging to a particular class and which are a subset of the `all[]` collection for this object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Collection` object	
JavaScript syntax:	IE	*`mySelect`*`.tags(`*`aTagName`*`)`
Argument list:	*`aTagName`*	The name of a tag to be filtered

This is a technique that only works in MSIE. The `tags()` method is used on all manner of collections.

The collection is traversed and all objects are examined to see if they were created by an HTML tag that is the same as that specified in the argument.

The argument must always be specified in upper case and the resulting collection will contain all objects of that type selected from the receiving collection object.

You can then manipulate the sub-set collection in the normal way, accessing items within it by index or by other means.

See also:	`Collection.tags()`

Select.type (Property)

The type of select object.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`mySelect`*`.type`

Because the `Select` object is not simulated by an `Input` object but by a specific class of its own, it can support a non-standard behavior for the `type` property.

In this case, the type indicates the number of simultaneous selections within the popup.

The values for this property may be one of the following:

- `SELECT-ONE`
- `SELECT-MULTI`

Property attributes:

`ReadOnly`.

Select.value (Property)

The presently selected option value.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*mySelect*.value

This is the string value that is sent back to the web server when the form is submitted.

This is equivalent to:

mySelect.options[*mySelect.selectedIndex*].value

It is the value of the first or only item selected in the popup. It will be the value of the first item when multiple items have been selected.

See also:	`Input.value`

Selection object (Object/browser)

An object representing a user selection in the current window.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*mySelection* = *myDocument*.selection
Object properties:	type	
Object methods:	`clear(), createRange(), empty()`	

This object represents a portion of the document in the current window that is currently highlighted, having been selected by the user or by a script. The selection is operated on by means of a `TextRange` object. This object is created by calling a `createRange()` method on the `Selection` object. This step is necessary because a selection cannot by its very nature persist very long, so a `TextRange` object encapsulates its value into a persistent store so it can be operated on, even though the original selection may have been deselected.

In Netscape Navigator, an entirely different technique is used that involves the `document.getSelection()` method.

Because it is easy to deselect the highlighted text by clicking on some other active object in the page, you will need to access the selection inside an event handler that is triggered by the selection action itself. This might be done quite effectively in an `onSelectStart` handler.

Warnings:

- `Selection` objects do not appear to be functional on any version of MSIE for the Macintosh. This may be because the `TextRange` objects have not been mapped to the Macintosh cut and paste architecture.

See also:	`Document.getSelection()`, `Document.selection`, `Password.select()`, `TextRange` object

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`type`	-	3.0 +	-	4.0 +	-	-	ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
`clear()`	-	3.0 +	-	4.0 +	-	-	-
`createRange()`	-	3.0 +	-	4.0 +	-	-	-
`empty()`	-	3.0 +	-	4.0 +	-	-	-

selection.clear() (Method)

A method to clear the area selected by the user.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*mySelection*`.clear()`

The contects of the current selection are cleared. This is very tricky to do neatly and even harder to accomplish in a portable manner. It is probably better to use the more structured `innerHTML` and related properties of the DOM hierarchy to accomplish the effects you want.

selection.createRange() (Method)

A factory method for creating a text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`TextRange` object	
JavaScript syntax:	*myTextRange* = IE	*mySelection*`.createRange()`

You will need to create a `TextRange` to operate on a `Selection` object in the MSIE browser. Once you have the `TextRange` you can then use script code to manipulate the content.

See also:	`TextRange` object

selection.empty() (Method)

A means of emptying a selection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*mySelection*`.empty()`

This deselects the text and sets the `Selection` object's type to "`None`". The document content is unchanged and the only visible artifact is that the highlighted area returns to its normal unhighlighted state.

selection.type (Property)

A property containing a type for the selection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*mySelection*`.type`

The selection type is reflected here. There are only two reasonable settings for this in the context of a web browser (although there are potentially many other kinds of selection that are possible). The property should contain either `None` or `Text`.

You can build a conditional check into your handler which only does something if the type of the `Selection` object is set to "`Text`".

See also:	`Input.type`

Property attributes:

`ReadOnly`.

Selection statement (Definition)

A means of selecting one or other code block to be executed.

Selection statements provide a means of executing one of several possible blocks of code. The simplest is the `if( ... )` selector. The next most complex is the `if( ... ) ... else ...` selector.

A similar and related concept is the condition execution operator `?:` which is functionally very similar to an `if (...) ... else ...` selection.

The ECMAScript standard reserves the `switch`, `case` and `default` keywords at edition 2 and mandates that they be supported functionally at edition 3.

See also:	Conditionally execute (`?:`), `else ...`, `if( ... ) ...`, `if( ... ) ... else ...`, `switch( ... ) ... case: ... default: ...`

SelectorArray object (Object/browser)

A collection of style sheet rules.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*mySelectorArray* = *myStyleSheet*.rules
Object properties:	length	
Object methods:	`item()`	

This is sometimes referred to as a `rules` object which is not strictly true. It is often so named because it is referenced by the `rules` property of a stylesheet.

DOM level 2 describes this object as a `CSSRuleList` object. It implies it is a sub-class of the `Collection` object and therefore it supports the `item()` method.

See also:	`Collection` object, `rule` object, `rule.selectorText`, `StyleSheet.rules[]`

Property	JavaScript	JScript	N	IE	Opera	HTML	Notes
`length`	-	3.0 +	-	4.0 +	-	-	ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	HTML	Notes
`item()`	-	3.0 +	-	4.0 +	-	-	-

SelectorArray.length (Property)

A count of the number of rules in a style sheet.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myStyleSheet*`.rules.length`

Property attributes:

`ReadOnly`.

Refer to:

`Collection.length`

self (Property)

An alias for the `window.self` property.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`Window` object	
JavaScript syntax:	-	*myWindow*`.self`
	-	`self`
	-	`top`
	-	`window`

This property yields a reference to the current window in which a script is running. This means that the statement `self.close()` is effectively `window.close()`.

This is another name for the `window.window` property in this context. However, `self` is useful because you can build reusable scripts with it that can be used with a variety of object types and instances. Don't forget that this can also refer to a `Frame` as well as a `Window` since they are both represented by the `Window` object.

The `self` property can be used without the `window` prefix because it belongs to the global object in a web browser window. Using the `self` keyword makes no difference to the functionality of a script but it makes it easier to understand. For the same reason, you may want to use the `window` property in the same way.

See also:	`Frame` object, `UniversalBrowserAccess`, `UniversalBrowserWrite`, `Window` object, `Window.frame`, `Window.self`

Property attributes:

`ReadOnly`.

Semantic event (Definition)

An event that has been mapped to the DOM.

See also:	Error handler, Event propagation

Semi-colon (;) (Delimiter)

Semi-colon characters are used to mark the end of a statement.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
JavaScript syntax:	-	*aStatement*;
Argument list:	*aStatement*	A JavaScript statement

Semi-colon characters are used to mark the end of a statement, separating one from another.

JavaScript is somewhat forgiving and will place semi-colons into the script automatically as needed except in some rare cases. Refer to the discussion on Automatic Semi-colon Insertion for more details.

Placing two semi-colons one after the other indicates a `null` statement. A line terminator can separate them and an optional comment is also permitted.

See also:	Automatic semi-colon insertion, Empty statement (;), Expression statement, Line terminator, Statement, `var`

Cross-references:

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 2 – section – 12.3

ECMA 262 edition 2 – section – 12.4

ECMA 262 edition 3 – section – 12.2

ECMA 262 edition 3 – section – 12.3

ECMA 262 edition 3 – section – 12.4

Wrox *Instant JavaScript* – page 18

SendMail object (Object/NES)

An object that encapsulates an outgoing e-mail message.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
JavaScript syntax:	NES	*mySendMail* = SendMail
	NES	*mySendMail* = new SendMail()
Object properties:	Bcc, Body, Cc, constructor, ErrorsTo, From, Organization, prototype, ReplyTo, Smtpserver, Subject, To	
Object methods:	errorCode(), errorMessage(), send()	

This provides a way for the Netscape Enterprise Server to send e-mail messages as a result of a client request.

You create a new message handling object with the `SendMail()` constructor. Then you define where it is going to be sent, its subject matter and content by storing string values in its various properties.

Finally, you transmit the message via an SMTP server with the `send()` method.

Example code:

```
<SERVER>
mySendMail = new SendMail();
mySendMail.Smtpserver = "mailhost";
mySendMail.To = "someone@somewhere.com";
mySendMail.From = "me@here.com";
mySendMail.Subject = "A test message";
mySendMail.Body = "Some body text";
mySendMail.send();
</SERVER>
```

See also:	Netscape Enterprise Server, `SendMail(),unwatch(),watch()`

Property	JavaScript	JScript	NES	Notes
Bcc	1.2 +	-	3.0 +	-
Body	1.2 +	-	3.0 +	-
Cc	1.2 +	-	3.0 +	-
constructor	1.2 +	-	3.0 +	-
ErrorsTo	1.2 +	-	3.0 +	-
From	1.2 +	-	3.0 +	-
Organization	1.2 +	-	3.0 +	-
prototype	1.2 +	-	3.0 +	-
ReplyTo	1.2 +	-	3.0 +	-
Smtpserver	1.2 +	-	3.0 +	-
Subject	1.2 +	-	3.0 +	-
To	1.2 +	-	3.0 +	-

Method	JavaScript	JScript	NES	Notes
`errorCode()`	1.2 +	-	3.0 +	-
`errorMessage()`	1.2 +	-	3.0 +	-
`send()`	1.2 +	-	3.0 +	-

SendMail() (Constructor)

A constructor for creating objects that encapsulate an outgoing e-mail message.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`SendMail` object	
JavaScript syntax:	NES	`new SendMail()`

Use this as an alternative method of constructing instances by cloning objects rather than instantiating fresh ones from the owning class.

See also:	Netscape Enterprise Server, `SendMail` object

SendMail.Bcc (Property)

Defines the list of blind copy recipients.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`mySendMail.Bcc`

This property contains the value that is transmitted in the BCC: header of the outgoing e-mail.

The value assigned to this property should be a comma separated list of BCC: recipient e-mail addresses.

SendMail.Body (Property)

The body text for the e-mail message.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.Body`

This property contains the value that is transmitted in the body of the outgoing e-mail.

You can store any valid 7 bit ASCII text here. If you want to get really sneaky, you can construct a MIME type separator and embed some attachments or send a multi-part message.

SendMail.Cc (Property)

Defines a list of CC recipients.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.Cc`

This property contains the value that is transmitted in the CC: header of the outgoing e-mail.

The value assigned to this property should be a comma separated list of CC: recipient e-mail addresses.

SendMail.constructor (Property)

A constructor function for object instances.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`Function` object	
JavaScript syntax:	NES	*`mySendMail`*`.constructor`

The constructor is that of the built-in `SendMail` prototype object.

You can use this as one way of creating mail dispatchers although it is more popular to use the new `SendMail()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

SendMail.errorCode() (Method)

Retrieves an error code value after attempting to transmit.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.errorCode()`

If the message sending failed, then you should be able to retrieve an error code value with this method.

SendMail.errorMessage() (Method)

If there is an error, then you can obtain the message text with this method.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.errorMessage()`

Use this method after calling the `send()` method to retrieve any pending error message text.

SendMail.ErrorsTo (Property)

Defines the recipient of error e-mails if the message fails to arrive.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.ErrorsTo`

This property contains the value that is transmitted in the ErrorsTo: header of the outgoing e-mail.

The value assigned to this property should be a valid email address (or comma separated addresses).

If the receiving mail system experiences any problems during the onwards delivery to the recipient's mailbox (for example, if it doesn't exist or is full), then a message is reflected back to the e-mail address in the ErrorsTo: header.

SendMail.From (Property)

Defines the From address of the e-mail.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`mySendMail.From`

This property contains the value that is transmitted in the From: header of the outgoing e-mail.

The value assigned to this property should be a valid e-mail address. It is considered to be bad practice to forge From addresses in mail headers, and some mail systems will not allow you to do this and will override the settings you define for this value.

Whatever value ends up in the From: header, the recipient can reply to this address if they mean to. It makes sense for this to be a mail address that a human being will ultimately check although you may want to put in some automation to read incoming e-mails if you expect a lot of responses.

SendMail.Organization (Property)

A standard header to describe the organisation you belong to.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.Organization`

This property contains the value that is transmitted in the Organization: header of the outgoing e-mail.

The value assigned to this property can be any 7 bit ASCII string.

SendMail.prototype (Property)

The prototype for the `SendMail` object that can be used to extend the interface for all `SendMail` objects.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`SendMail` object	
JavaScript syntax:	NES	`SendMail.prototype`
	NES	*`mySendMail`*`.constructor.prototype`

Refer to:

`prototype` property

SendMail.ReplyTo (Property)

The address to which replies should be sent.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.ReplyTo`

This property contains the value that is transmitted in the ReplyTo: header of the outgoing e-mail.

The value assigned to this property should be a comma separated list of ReplyTo: recipient e-mail addresses.

Beware that many mail clients do not honor this field, they will attempt to send messages back to the ErrorsTo: or From: addresses or may even make one up. These header values are merely conventions and although they are described in various RFC documents, mail clients are written by people whose skills and attention to detail range from the plain careless to the highly professional.

SendMail.send() (Method)

Sends the message encapsulated by this object.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
JavaScript syntax:	NES	`mySendMail.send()`

This connects to the `sendmail` process in the server and dispatches the message. There is an implication here that because it uses `sendmail`, then the server can only run on platforms that support `sendmail`. That further implies the platform is a Unix system, although `sendmail` is open source and may be deployed on non-Unix platforms.

If `sendmail` is not available the results of calling this method are uncertain.

SendMail.Smtpserver (Property)

Defines the name of the SMTP server to which we shall connect and request that our message be sent.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.Smptpserver`

You will need to consult your system administrator about this value. It is impossible to state with any certainty what value you should assign to this property there are so many possible ways to configure a mail server and deploy it. It may or may not be on the same machine as your web server.

SendMail.Subject (Property)

Defines the subject heading for the message.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.Subject`

This property contains the value that is transmitted in the Subject: header of the outgoing e-mail.

The value assigned to this property should be a 7 bit ASCII string.

SendMail.To (Property)

Defines the To: address for the e-mail.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	*`mySendMail`*`.To`

This property contains the value that is transmitted in the To: header of the outgoing e-mail.

The value assigned to this property should be a comma separated list of To: recipient e-mail addresses.

These are the recipients of the e-mail and if you specify several, it may be more efficient as your SMTP server can bulk deliver several messages at once.

server object (Object/NES)

An object that represents the server in server-side JavaScript implementations.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
JavaScript syntax:	NES	`server`
Object properties:	`agent, host, hostname, port, protocol`	
Object methods:	`lock(), unlock()`	

This is a server-side host object representing the server. There is only one and you cannot instantiate it although you can make references to it. All users on the web server share this object.

This is an object that allows you to share values across all sessions running in all applications across the entire server. The locking facilities permit you to lock resources while you are using them.

Because this applies server-wide, there is even more reason to ensure you lock objects for the minimum of time and relinquish the locks as soon as possible. It is quite feasible to completely stall the whole server by locking a vital resource during the processing of a single client request. The effect of this is to make your server a single-threaded non-concurrent session server. That is, it will only actually serve one client request at a time.

See also:	Netscape Enterprise Server, `project` object, `response.server`, `unwatch(),watch()`

Property	JavaScript	JScript	NES	Notes
`agent`	1.1 +	-	2.0 +	-
`host`	1.1 +	-	2.0 +	-
`hostname`	1.1 +	-	2.0 +	-
`port`	1.1 +	-	2.0 +	-
`protocol`	1.1 +	-	2.0 +	-

Method	JavaScript	JScript	NES	Notes
`lock()`	1.1 +	-	2.0 +	Warning
`unlock()`	1.1 +	-	2.0 +	-

Cross-references:

Wrox *Instant JavaScript* – page 65

Wrox *Instant JavaScript* – page 67

server.agent (Property)

Describes the server being used.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`server.agent`

The `userAgent` property of the navigator object in a web browser describes the kind of browser being used. This value is presented as a text string.

This is the corresponding property in a web server and provides a way to deploy scripts that are shared amongst several variants of a server that are compatible but different.

See also:	`Navigator.userAgent`

server.host (Property)

The machine and domain name values for the server host.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`server.host`

This value will be the DNS name that you can use to reach the server.

You might use this value to manufacture URL values for use in `HREF="..."` HTML tag attributes for example.

See also:	Netscape Enterprise Server, `unwatch(),watch()`

server.hostname (Property)

The hostname and port property of the receiving server object.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`server.hostname`

This value will be the same as the `host` property if the port number the server listens on is 80. Otherwise, the port number will be appended to the `host` value and that will be the `hostname` value. A colon separator is introduced between the host and port values.

See also:	Netscape Enterprise Server

server.lock() (Method)

A means of locking the server object to prevent contention between scripts.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
JavaScript syntax:	NES	`server.lock()`

The server lock would be used at a higher hierarchical level than the project lock mechanism.

It is vitally important that you don't hog a lock on the server object as this can deny server access to all other users

The lock will stall if another script currently has a lock extant on this project. The method will then return when that lock is relinquished.

Warnings:

- Locking the server object can lead to severe performance degradation. While it is locked, any lock requests made by other scripts will stall pending the lock being relinquished with an `unlock()` method call.
- You can render your server virtually useless by over-locking the `server` object. You should aim to relinquish any locks as soon as you possibly can. Ideally you should seek to avoid locking the server object if at all possible.

See also:	`project.lock()`

server.port (Property)

The port number that the server listens on for incoming requests.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	NES	`server.port`

This value is normally port number 80, but ports 81, 8080 or 8081 are also commonly used. It can be any port number you like although there are many reserved ports for other services on the host. On a Unix system, there is an `/etc/services` file that will enumerate the ports that are likely to be reserved for other non web-server purposes.

You may also have difficulty in using ports below number 1024 unless you have access to the system administrator account.

server.protocol (Property)

The protocol supported by the server is available from this property.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	NES	`server.protocol`

Typical values for this property will be "`http:`" or "`https:`". The latter will be used if the server is a secure server providing e-commerce support for example.

server.unlock() (Method)

Relinquisesh a lock on the server object.

Availability:	JavaScript – 1.1 Netscape Enterprise Server version – 2.0	
JavaScript syntax:	NES	`server.unlock()`

If you ever lock the server object from a script, you should call this as soon as you possibly can after you have completed your lock critical code. Retaining a lock on your `server` object is not only bad manners but is seriously bad for the health and well-being of your server. Locking `server` objects persistently will hog resources and your server performance will slow to a crawl.

See also:	`project.unlock()`

Server side browser detection (Useful tip)

You can do a great deal of browser portability handling if you are prepared to serve browser specific pages from your web server.

A wholly dynamic site may be able to serve browser, platform and version specific HTML according to the value of the `userAgent` string that the browser sends when it makes a request. This technique is fine for a few pages and when they don't experience high traffic. To serve static pages this way needs a smarter server-side trick to be deployed.

You could render very browser specific copies of your pages and store them under document path names that contain a component that could be derived by disassembling the `userAgent` string. Then in your web server, you can trap every request that needs this capability, route it through a special module and generate a browser-specific path modification. If you do this creatively, you could provide a mechanism that allows the web pages to request an unmodified URL but the web server serves a page that is as close to ideal as it can get.

For example, the browser might be Netscape 3 on a Macintosh. That might yield a string containing a `3` for the version, an `N` for the browser and an `M` for the platform. The `3` might become a `4` or even a `6` for other versions of Netscape Navigator. The `N` might become an `E` for MSIE, an `I` for iCab or an `O` for Opera. The `M` might become a `W` for windows. So we have a string that represents the browser, platform and version in three letters. We might get a string such as `NM3`, for example. When the browser requests a page called `index.html`, the web server would inspect the `userAgent` string and work out that its normalized signature is `NM3`. The web server can then attempt to serve the page `NM3index.html`. If this does not exist, the web server can fall back to `NMindex.html` and then `Nindex.html` before eventually serving just plain old `index.html`. These tests in the web server will take fractions of a second with something like a `stat()` function call to test for the existence of a file.

With this technique, your publishing logic can generate some very platform, browser and version specific static files and the web server can locate them quickly and efficiently even when there are high traffic loads on the server farm. It's also workable for style sheets and can be deployed in a load-balanced multi-machine server farm as well.

See also:	Compatibility

Server-side JavaScript (Definition)

That JavaScript which is executed in the web server, probably in response to a browser request and accessed via CGI.

Server-side JavaScript is when the JavaScript source is executed in response to a request from a user's client application. That request arrives as a server, which then determines that a JavaScript needs to be executed to build the response. The server then runs the script and returns the output of it to the user, most likely in the form of a web page but possibly in the form of some image data or other textual or binary content.

Some server-side implementations are designed to yield performance improvements by semi-compiling the JavaScript and retaining that byte-code form in a cache.

Each server may offer alternative ways to invoke the server-side scripting. It is hoped that all the servers that use JavaScript and which conform to ECMAScript may well use the same tags for enclosing the server-side script. However, since the tags are not defined in the standard, this may be a vain hope.

See also:	Active Server Pages, CGI Driven JavaScript, Desktop JavaScript, Internet Information Server, LiveWire, Netscape Enterprise Server, Shell Scripting with JavaScript

Cross-references:

Wrox *Instant JavaScript* – page 3

Wrox *Instant JavaScript* – page 5

Wrox *Instant JavaScript* – page 64

Wrox *Professional JavaScript* – page 59-90

setHotkeys() (Method)

Activate or deactivate keyboard shortcuts for this window.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myWindow*`.setHotKeys(`*aSwitch*`)`
	N	`setHotKeys(`*aSwitch*`)`
Argument list:	*aSwitch*	A Boolean switch value

Refer to:

`Window.setHotkeys()`

setInterval() (Method)

A method for setting timer intervals.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	- `myWindow.setInterval(aFunction, anInterval, someArguments)`	
	- `myWindow.setInterval(aSourceText, anInterval)`	
	- `setInterval(aFunction, anInterval, someArguments)`	
	- `setInterval(aSourceText, anInterval)`	
Argument list:	*aFunction*	A function object (not supported in MSIE)
	anInterval	A time interval in milliseconds
	aSourceText	Some valid JavaScript source text
	someArguments	The arguments to the function object (not supported in MSIE)

See also:	Timer events, `Window.clearInterval()`, `Window.setTimeout()`, `Window.setInterval()`

setResizable() (Method)

Enable or inhibit the window resize capability.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`myWindow.setResizable(aSwitch)`
	N	`setResizable(aSwitch)`
Argument list:	*aSwitch*	A Boolean value to control the functionality

Refer to:

`Window.setResizable()`

setTimeout() (Method)

A method for setting a one shot timer.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`myWindow.setTimeout(`*`aSourceText, aWaitTime`*`)`
	-	`setTimeout(`*`aSourceText, aWaitTime`*`)`
Argument list:	*`aSourceText`*	Some valid JavaScript source text
	`aWaitTime`	A delay in milliseconds

See also:	Timer events, `Window.setInterval()`, `Window.setTimeout()`

setZOptions() (Method)

Defines the window stacking behaviour.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myWindow`*`.setZOptions(`*`anOptionValue`*`)`
	N	`setZOptions(`*`anOptionValue`*`)`
Argument list:	*`anOptionValue`*	One of a range of possible settings for the feature

Refer to:

`Window.setZOptions()`

Shadow() (Filter/visual)

A visual filter for creating a shadow.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Filter – `Shadow()`

Shallow copying (Definition)

Copying object references and not the objects.

See also:	`Array.toSource()`

Refer to:

Copying objects

Shared Property (Definition)

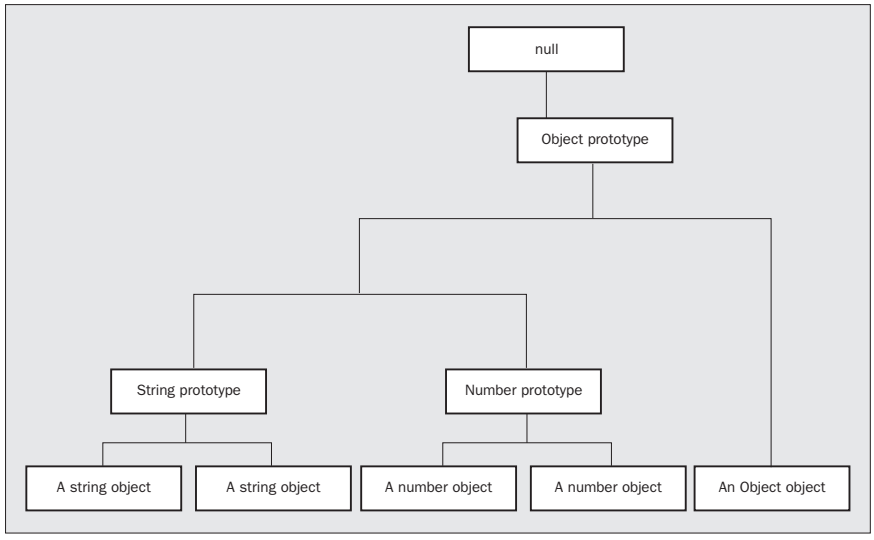
A property contained in a prototype and shared between several instances.

Availability:	ECMAScript edition – 2

You can create several objects and make them share a common ancestor. In a class-based object oriented world, this would be called sub-classing. Instantiating each one means it would inherit properties from its super-class, but objects of the same class in a real object oriented system do not share property values unless the static (class) factory method that instantiates them presets the same values as they are initialized.

In JavaScript, because the prototype chain is used to inherit properties from parent objects and not parent classes, then objects will inherit property values unless they override them locally.

If you are used to the class-based object oriented way of doing things, this can be quite distracting.



See also:	Prototype Based Inheritance, Prototype chain, `prototype` property

Cross-references:

ECMA 262 edition 2 – section – 10.1.4

Shell Scripting with JavaScript (Definition)

Unix command line tools written in JavaScript.

Once you have a JavaScript interpreter installed on your system, possibly to provide a server-side or CGI handling mechanism, you can easily use it then to do the things you might previously have done in Perl, Tcl, Bourne, Korn or C-Shell.

Not that you would necessarily choose JavaScript over the other alternatives but for some projects it might be better suited due to the kind of data you are manipulating or what you need to do to it.

The main strength of the Unix environment is the way multiple scripting languages can be used at will and a project may be built from code that runs in many different environments and contexts.

See also:	CGI Driven JavaScript, Host environment, Platform, Server-side JavaScript

Cross-references:

Wrox *Instant JavaScript* – page 5

Shift expression (Definition)

Shifts the left value by an amount specified by the right value.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

Bitwise shift operators convert their left operands to a 32-bit integer value and shift them according to their right operation. The operator determines the kind of shifting that is applied.

See also:	Bitwise shift operator, Expression

Cross-references:

ECMA 262 edition 2 – section – 11.7

ECMA 262 edition 3 – section – 11.7

Shift operator (Definition)

Used to create a shift expression.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

Shift operators convert their left operands to a 32-bit integer value and shift them according to their right operation. The operator determines the kind of shifting that is applied.

See also:	Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=)

Cross-references:

ECMA 262 edition 2 – section – 11.7

ECMA 262 edition 3 – section – 11.7

short (Reserved word)

Reserved for future language enhancements.

The inclusion of this reserved keyword in the ECMAScript standard suggests that future versions of ECMAScript may be more strongly typed.

This keyword also represents a Java data type and the `short` keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

See also:	double, float, Integer, LiveConnect, long, Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

showHelp() (Method)

Displays the help window.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myWindow*.showHelp()
	IE	showHelp()

Refer to:

Window.showHelp()

showModalDialog() (Method)

An alias for the `window.showModalDialog()` method.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	User defined	
JavaScript syntax:	IE	*myWindow*.showModalDialog(*aURL*, *someArguments*)
	IE	showModalDialog(*aURL*, *someArguments*)
Argument list:	*aURL*	A URL to load into the modal dialog
	someArguments	Arguments to pass to the modal dialog

Refer to:

`Window.showModalDialog()`

showModelessDialog() (Method)

An alias for the `window.showModelessDialog()` method.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	User defined	
JavaScript syntax:	IE	*myWindow*.showModelessDialog(*aURL*, *someArguments*)
	IE	showModelessDialog(*aURL*, *someArguments*)
Argument list:	*aURL*	A URL to load into the modal dialog
	someArguments	Arguments to pass to the modal dialog

Refer to:

`Window.showModelessDialog()`

.shtm (File extension)

Server-side processed HTML file.

Refer to:

File extensions

.shtml (File extension)

Server-side processed HTML file.

Refer to:

File extensions

Side effect (Definition)

The changes to the execution environment due to some code being executed.

When functions are executed, they may alter some item in memory or cause some change to occur aside from simply returning a value. For example, a function may create an object and add it to a persistent array and return some value such as the number of elements in that array. The calling expression was not aware that the array was extended and simply received a numeric value as a result of evaluating the function. The side effect was that the array became longer by one element.

See also:	Expression, Script execution

Sidebar object (Object/Navigator)

A new object introduced with Netscape 6.0 to manage the left side navigation bar.

Availability:	JavaScript – 1.5 Netscape – 6.0	
JavaScript syntax:	N	*myWindow*.sidebar
	N	sidebar

This is a new object which needs to be explored as we get to know the netscape Navigator 6.0 browser. It encapsulates the behavior and appearance of the sidebar frame on the left of the browser window.

See also:	`Window.sidebar`

Signed scripts (Security related)

A means of giving scripts a privilege to access secure content.

Netscape Navigator allows scripts to have digital signatures attached to them. These signatures can control the level of privilege that a script is allowed to have in a web browser. This is ultimately under user control but if the user allows, the scripts can be secured at source.

The signature combines the identity of the signatory and a checksum of the content. The content cannot be modified without invalidating the checksum and hence voiding the signature.

It would be difficult to establish the exact security criteria beforehand, so Netscape Navigator forces scripts to request the privileges they need. Then, you can allow or deny the access which can be stored and mapped against the identity of the person signing the script. This means a security policy can gradually be established by training the browser to recognize and make decisions on access. Initially, no access is available but after some time, your browser preferences will contain a very sophisticated set of rules that govern the access to the secure values.

To sign your scripts, you will need additional tools and utilities. These are available from Netscape and should form part of your publishing pipeline.

An alternative is to serve your scripts separately from a secure server. Scripts served in this way will assumed to have been signed by the secure server itself.

As a way round the inconvenience of signing scripts after every minor correction, you can sign the codebase of a script. This means you can establish a security setting for scripts from a specific web server. It is slightly less secure than signing a checksum but more convenient during development. It is recommended that proper signing be used; once the script changes are less frequent and the development process is complete, the scripts will be more stable and signing will be carried out less frequently. With some automation in the publishing work flow, you may be able to sign scripts as part of the releasing procedure that your developers employ.

Your web page may contain more than one script. For signed script access control to work, all of the scripts on a page must be signed. If an unsigned script is present, it defeats the entire signing status of the whole page. Scripts can be signed by more than one person. Netscape Navigator will try and find the highest most complete coverage of the scripts in a page. Ideally it will find a particular signer who has signed all of the scripts. Other signers may have conferred a higher level of security but not on all of the scripts. The more complete coverage will prevail.

Although these fairly strict same-signer policies apply to the scripts within a window, scripts in multiple windows may operate under a slightly relaxed policy. The "same signer" policy is a variation of the "same origin" policy. Different signers cause the browser to behave as if the pages were from different origins. Both scripts may have rights to request `UniversalBrowserRead` access which might work around the problem.

Unsigned scripts have quite restricted access to window properties for windows that contain signed scripts. This means that untrusted and insecure scripts cannot access secure data by subverting an already trusted script.

Warnings:

- MSIE version 4 does not support the Netscape Navigator privilege model. Therefore scripts are always unprivileged. The MSIE security model is based on zones. This is a fairly coarse grained approach and simply allows scripts to be executed or not as a whole. The Netscape Navigator model allows access to be controlled object by object.
- Being so closely related to the MSIE browser, the WebTV box also does not support signed scripts.

See also:	AuthentiCode, Code signing, Data-tainting, `export`, `import`, JellyScript, Requesting privileges, Same origin

Web-references:

http://developer.netscape.com/software/signedobj/
http://developer.netscape.com/library/documentation/signedobj/signtool/

Single line comment (Definition)

A pair of slash characters (`//`) indicates single line comments.

Single line comments are indicated by a pair of slash characters (`//`) and are completed by a line terminator.

The pair of slash characters and everything following them to the end of the line is considered to be a comment. Comments are discarded during the interpretation phase of a line of script. A pair of slash characters would not behave as a comment delimiter if one of them were escaped with a backslash character or if they appeared inside a single or double quoted string literal.

See also:	Comment, Comment (`//` and `/* ... */`), Line

Cross-references:

ECMA 262 edition 2 – section – 7.3

ECMA 262 edition 3 – section – 7.4

Wrox *Instant JavaScript* – page – 17

Slide() (Filter/transition)

A transition effect with the appearance of one image sliding over another.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

Filter – `Slide()`

SMALL object (Object/HTML)

An object that represents the font style controlled by the `<SMALL>` HTML tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Deprecated:	Yes	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`mySMALL = myDocument.all.anElementID`
	IE	*`mySMALL`* `=` *`myDocument`*`.all.tags("SMALL")[`*`anIndex`*`]`
	IE	*`mySMALL`* `=` *`myDocument`*`.all[`*`aName`*`]`
	-	*`mySMALL`* `=` *`myDocument`*`.getElementById(`*`anElementID`*`)`
	-	*`mySMALL`* `=` *`myDocument`*`.getElementsByName(`*`aName`*`)[`*`anIndex`*`]`
	-	*`mySMALL`* `=` *`myDocument`*`.getElementsByTagName("SMALL")[`*`anIndex`*`]`
HTML syntax:	`<SMALL> ... </SMALL>`	
Argument list:	*`anIndex`*	A reference to an element in a collection
	`aName`	An associative array reference
	`anElementID`	The `ID` value of an `Element` object
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

snews: URL (Request method)

A request from a web browser to a secure news server to send a document.

Use the browser to download and browse some content from a secure news site.

See also:	javascript: URL, Security policy, URL

Sort ordering (Definition)

The mechanism by which items are arranged in sequence according to a locale.

See also:	Localization

Refer to:

Collation sequence

Source files (Definition)

You can store JavaScript into external source files.

You can link JavaScript source files at the client end, which causes the browser to fetch them and bind them into the page just like any other asset such as a style-sheet or image.

See also:	`.js`

Cross-references:

Wrox *Instant JavaScript* – page – 65

Wrox *Professional JavaScript* – page – 103

Source text (Definition)

Human readable script source text to be parsed and executed.

Availability:	ECMAScript edition – 2
See also:	Comment, Escape sequence (\), Lexical convention, Script

Refer to:

Script source text

Cross-references:

ECMA 262 edition 2 – section 6

ECMA 262 edition 3 – section 6

SPAN object (Object/HTML)

An object that encapsulates the contents of an inline `<SPAN>` tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*mySPAN* = *myDocument*.all.*anElementID*
	IE	*mySPAN* = *myDocument*.all.tags("SPAN")[*anIndex*]
	IE	*mySPAN* = *myDocument*.all[*aName*]
	-	*mySPAN* = *myDocument*.getElementById(*anElementID*)
	-	*mySPAN* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*mySPAN* = *myDocument*.getElementsByTagName("SPAN")[*anIndex*]
HTML syntax:	`<SPAN> ... </SPAN>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`dataFld, dataFormatAs, dataSrc`	
Event handlers:	`onBlur, onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

<SPAN> tags and the objects that represent them are inline elements. Placing them into a document does not create a line break.

Note that a positioned <SPAN> elements will appear as a member of the document.layers[] collection in Netscape 4.

The example shows how properties of <SPAN> blocks can be moved from one to another. In this example, the background color of each block is moved along to the next in a cyclic manner as the mouse is clicked:

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SPAN ID="AAA" STYLE="background-color:RED">ONE</SPAN>
<SPAN ID="BBB" STYLE="background-color:BLUE">TWO</SPAN>
<SPAN ID="CCC" STYLE="background-color:GREEN">THREE</SPAN>
<SPAN ID="DDD" STYLE="background-color:CYAN">FOUR</SPAN>
<SPAN ID="EEE" STYLE="background-color:YELLOW">FIVE</SPAN>
<SPAN ID="FFF" STYLE="background-color:GRAY">SIX</SPAN>
<FORM>
<INPUT TYPE="button" VALUE="CLICK ME" onClick="clickMe()">
</FORM>
<SCRIPT>
//IE only
function clickMe()
{
   mySpans = document.all.tags("SPAN");
   myStyle1 = mySpans[mySpans.length-1].style.cssText;

   for(myEnum=0; myEnum<mySpans.length; myEnum++)
   {
      myStyle2 = mySpans[myEnum].style.cssText;
      mySpans[myEnum].style.cssText = myStyle1;
      myStyle1 = myStyle2;
   }
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Element object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
dataFld	-	3.0 +	-	4.0 +	-	-	-	-
dataFormatAs	-	3.0 +	-	4.0 +	-	-	-	-
dataSrc	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onBlur	-	3.0 +	-	4.0 +	-	-	-	Warning
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Special number values (Definition)

Special properties of the `Global` and `Number` objects.

Special number values are provided so that scripts can test for exceptional values yielded as a result of arithmetic expressions.

You can check for infinity or non-number errors as a result of a divide by zero for example. Maximum and minimum values are also available to range check input data to make sure it can be used in arithmetic computations.

You cannot assign constant values of `Infinity` or `NaN` because you cannot type in a constant value to denote them. However because they exist as properties of the `Global` object, you can refer to them by name.

Warnings:

- Be careful that you don't accidentally assign new values to the `Infinity` and `NaN` properties of the global object. Some implementations do not properly protect them against being written to and your script may assign new values to them. This can lead to very unpredictable behavior.

See also:	`Infinity`, `NaN`, `Number.MAX_VALUE`, `Number.MIN_VALUE`, `Number.NaN`, `Number.NEGATIVE_INFINITY`, `Number.POSITIVE_INFINITY`

Cross-references:

Wrox *Instant JavaScript* – page 14

Special type (Definition)

Special data types are available to test variable content.

There are two special data types available to test references to objects or the contents of variables. If a variable is currently of the type `undefined`, then it has not had any value stored in it. If it is a property, perhaps it has been deleted from the object.

A variable reference would also yield the undefined value if it has not been declared. However, a reference to an undeclared variable causes a run-time error.

Properties will yield the value `null` if they are supposed to contain an object and have been purposely nulled out. They will be `undefined` unless they have been set to `null` by an assignment.

You can examine variables and object properties with the enquiry function to determine the type of the value stored there. The enquiry functions are:

- `isNaN()`
- `isFinite()`

This operator behaves like an enquiry function:

- `typeof`

Testing for `null` and `undefined` or comparing them one with another can be problematical and is certainly non-trivial. In older versions of the browsers, you can simulate missing values.

The `null` value can be simulated with this expression:

```
(void 0)
```

The `undefined` value can be simulated with this:

```
(void null)
```

Of course the simulations depend on the existence of some keywords that may or may not exist.

See also:	`isFinite(),isNaN(),null,typeof,undefined type`

Spiral() (Filter/transition)

Reveals the new image with a spiral effect.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

Filter – `Spiral()`

SSJS (Definition)

An abbreviation for Server-Side JavaScript.

Refer to:

Server-side JavaScript

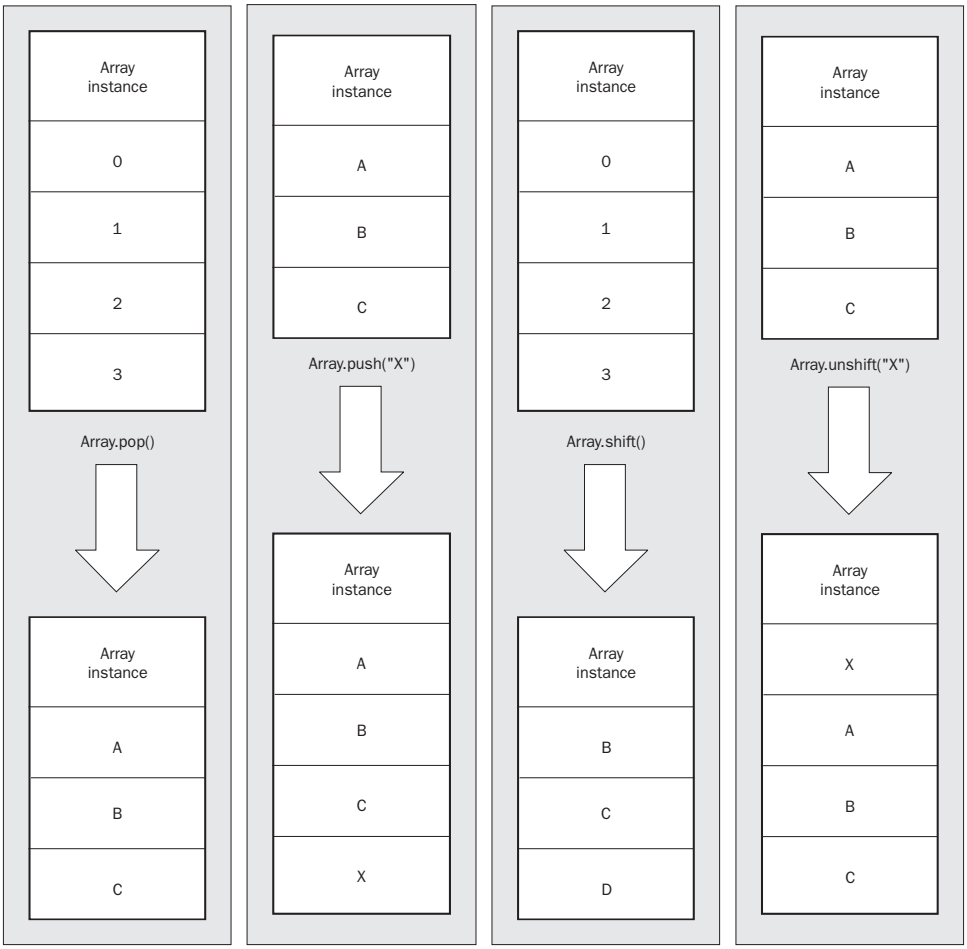
Stack manipulation (Useful tip)

In Netscape 4, you can build stack managers.

With the `Array` methods `push()`, `pop()`, `unshift()` and `shift()`, you can build various stacks and queues.

A First In Last Out stack can be constructed with `push()` and `pop()`. This works from the end of the array.

An alternative FILO stack can be constructed with `unshift()` and `shift()` but that will operate at the start of the array.



See also:	`Array.pop()`,`Array.push()`,`Array.shift()`, `Array.unshift()`,Queue manipulation

Standalone JavaScript (Definition)

JavaScript that is executed in an application not associated with web pages at all.

A stand-alone JavaScript implementation is when the interpreter is embedded into an application other than a web browser or web server. The application may utilize JavaScript as a mechanism for connecting its user interface to its internal functionality. This gives some benefits in being able to custom-tailor the way an application behaves and what is really delivered is a kit of components that are joined up with JavaScript glue.

Stand-alone interpreters may have the scripts embedded inside the application or they may read them in from external sources.

An example of a well known stand-alone interpreter is the Nombas ScriptEase range of JavaScript interpreters. The ScriptEase implementations have many additional objects to provide a very rich JavaScript programming environment indeed.

See also:	Nombas ScriptEase

Cross-references:

Wrox *Instant JavaScript* – page 68

Statement (Definition)

A functional section of a program.

Availability:	ECMAScript edition – 2

A statement is a discrete instruction in a script that causes something to happen.

The statements in JavaScript can be classified into several categories. Here are the basic set of classifications:

- Block – Some code enclosed in braces.
- Variable statement – A declaration of a local or global variable.
- Empty statement – A semi-colon on its own.
- Expression statement – An operator and its require operand(s).
- If statement – Conditional execution of a block of code with an optional alternative block of code.
- Iterative statement – A means of executing a block of code repetitively until a test condition is satisfied.
- Switch selector – A means of executing one of a variety of possible code blocks selecting the best according to an input value.
- Continue statement – A way of cancelling an iteration and commencing the next.

- Break statement – Means of breaking out of an iteration or a switch selector.
- Return statement – A way to unconditionally leave a function and return to its caller, optionally handing back a value.
- With statement – A means of adding an object to a scope chain.

Statements are executed in the order in which they appear in the script source text except when the flow of control is redirected by a conditional switching expression, function call, iterator or jump statement.

See also:	`break`, Compound statement, continue, Empty statement (`;`), Expression statement, `if( ... ) ...`, `if( ... ) ... else ...`, Iteration statement, JavaScript language, Jump statement, Method, Punctuator, `return`, Script fragment, Semi-colon (`;`), `var`, Variable statement, `with ...`

Cross-references:

ECMA 262 edition 2 – section 12

ECMA 262 edition 3 – section 12

Wrox *Instant JavaScript* – page 16

static (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Static filters (Definition)

These filters are used to define the appearance of an HTML `Element` object.

They are all members of the visual filter family and are invoked through the stylesheet mechanisms or by the style object's filter property. Here is a list of the static filters:

- `Alpha()`
- `BasicImage()`
- `Blur()`
- `Chroma()`
- `Compositor()`
- `DropShadow()`
- `Emboss()`
- `Engrave()`

- `FlipH()`
- `FlipV()`
- `Glow()`
- `Grayscale()`
- `Invert()`
- `Light()`
- `Mask()`
- `MaskFilter()`
- `Matrix()`
- `MotionBlur()`
- `Pixelate()`
- `Shadow()`
- `Wave()`
- `XRay()`

Some of these may not be available with all versions of the MSIE browser. Refer to the individual topics for availability.

See also:	`style.filter`, Visual filters

Static method (Definition)

Defines static methods using function properties you add to a function object.

Static methods are also known as class methods. These are associated with a class rather than an instance of a class. The constructor function is analogous to the factory class for objects in truly object oriented systems.

Static methods have no real use for the `this` keyword, as it is meant to refer to the receiving instance.

These are called static methods because they are not associated with a local instance.

See also:	Function object properties

Static variable (Useful tip)

Defines static variables using properties you add to a function object.

Static variables are also known as class variables.

If we want to create some class variables, we need to add properties to the constructor function object.

If we have made our own constructor function, we can add our own class variables to it.

We can also add class variables to the built-in objects in some implementations.

These are called static variables because they are not associated with a local instance.

See also:	Function object properties

status (Property)

An alias for the `window.status` property.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myWindow*`.status`
	-	*myWindow*`.status = `*aString*
	-	`status`
	-	`status = `*aString*
Argument list:	*aString*	A string to display in the status bar

See also:	`Window.defaultStatus, Window.status`

Status code (Result value/NES)

Many of the NES supported methods return a status code that is consistently defined across all objects and methods.

The following methods will return a status code:

- `database.execute()`
- `database.beginTransaction()`
- `database.commitTransaction()`
- `database.rollbackTransaction()`
- `cursor.insertRow()`
- `cursor.updateRow()`
- `cursor.deleteRow()`

The status codes are summarized in the table:

Code	Meaning
00	No error
01	Out of memory
02	Object was never initialized
03	Type conversion error
04	Database not registered
05	Error reported by database engine
06	Message from database engine
07	Error from database vendor's library
08	Lost connection
09	End of fetch
10	Invalid use of object
11	Column does not exist
12	Bounds error – invalid positioning within object
13	Unsupported feature
14	Null reference parameter
15	Database object not found
16	Required information missing
17	Object cannot support multiple readers
18	Object cannot support deletes
19	Object cannot support inserts
20	Object cannot support updates (1)
21	Object cannot support updates (2)
22	Object cannot support indices
23	Object cannot be dropped
24	Incorrect connection supplied
25	Object cannot support privileges
26	Object cannot support cursors
27	Unable to open

Status codes 5 and 7 are significant. It depends on the database being used as to which is important.

Status code 5 is important for Oracle and ODBC.

Status code 7 is important for Informix and Sybase.

If these values are detected, then the major and minor error codes and messages can be inspected for further help in diagnosing the problems.

See also:	Connection.majorErrorCode(),Connection.majorErrorMessage(), Connection.minorErrorCode(),Connection.minorErrorMessage(), Cursor.deleteRow(),Cursor.insertRow(),Cursor.updateRow(), database.beginTransaction(),database.commitTransaction(), database.execute(),database.majorErrorCode(), database.majorErrorMessage(),database.minorErrorCode(), database.minorErrorMessage(),database.rollbackTransaction(), Error handling

Status line (Definition)

An area in the browser frame that status messages can be presented in.

When browsers open a new window, they will have a status line at the bottom (unless you choose to deactivate the status line).

This can have text messages presented to tell the user what is happening. One particularly popular usage is to present rollover text messages associated with links.

There are two property values on each window that control the content of this status line.

The `defaultStatus` property contains the text that is displayed when the mouse is not rolled over an active item.

The `status` property is set to a specific value by an `onMouseOver` event handler and resets when the mouse leaves.

Warnings:

- The correct behavior is for the `onMouseOver` event handler to override the default behavior and replace the text in the status bar. Then, when the mouse rolls off of the active item, the defaultStatus value should be restored automatically.
- This is not the case with all browsers. Macintosh and X-Windows versions of Netscape 3 (and possibly other browser versions) do not automatically restore the previous status line value. You will need to add a `onMouseOut` handler to explicitly reset the status line.

See also:	onMouseOut, onMouseOver, Window.defaultStatus, Window.status

statusbar (Property)

An alias for the `window.statusbar` property.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Bar` object	
JavaScript syntax:	-	*myWindow*.statusbar
	-	statusbar

See also:	`Bar` object, `Window.statusbar`

Property attributes:

`ReadOnly`.

.stm (File extension)

Server-side processed HTML file.

Refer to:

File extensions

stop() (Method)

An alias for the `window.stop()` method.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`myWindow.stop()`
	N	`stop()`

Refer to:

`Window.stop()`

Storage duration (Definition)

The time during which an entity is available for use.

Objects and other entities may be created and destroyed at will while a script is being executed.

Some are created automatically and some also get destroyed automatically. Some will be discarded when the script terminates either intentionally or accidentally due to an error of some kind.

Objects may only persist while a function is being executed. They may vanish when the function returns to its caller.

In a web browser, generally speaking, objects persist for the life of a page. They will be destroyed when the page is discarded or refreshed. Session storage can be accomplished effectively by maintaining a frame set where the outermost frame remains available throughout the session even though the pages it contains are replaced several times.

In a server-side environment, objects and entities will likely only persist during the request-response loop. Items may persist longer if the server-side implementation is able to archive them or if it is able to maintain session state information between one request and another.

In a TV set-top box environment, objects may persist for some time, perhaps for the duration of a broadcast program or for as long as the TV set-top box is tuned to a particular channel. Changing channels may purge out the object store. This particular kind of implementation is under constant and rapid development. One of the major areas of research is that of persistent and browsable cache systems where objects may repose and be recalled at will. In a system like that, objects may persist forever, or until the user chooses to dispose of them explicitly.

See also:	Identifier, Request-response loop, Scope chain, Script execution, Script termination

Stproc object (Object/NES)

An object that encapsulates a call to a stored procedure on a database from a Netscape Enterprise Server.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
JavaScript syntax:	NES	`myStproc = database.storedProc(aProcName, aProcParm);`
	NES	`myStproc = myConnection.storedProc(aProcName, aProcParm)`
Argument list:	`aProcName`	The name of a stored procedure to call
	`aProcParm`	A parameter value to pass to the stored procedure
Object properties:	prototype	
Object methods:	`close(), outParamCount(), outParameters(), resultSet(), returnValue()`	

This object provides a container to manage the call to the stored procedure and somewhere that the results can be made available for further processing by your scripts.

You create `Stproc` objects by requesting them from the database or connection objects that are accessing the target database you are interested in.

Example code:

```
<SERVER>
// An example derived from Wrox Professional JavaScript
database.connect("ODBC", "myDatabase", "me", "myPassword", "");
myStproc = database.storedProc("myProcedure", 40);
</SERVER>
```

See also:	Connection.storedProc(), Netscape Enterprise Server

Property	JavaScript	JScript	NES	Notes
prototype	1.2 +	-	3.0 +	-

Method	JavaScript	JScript	NES	Notes
close()	1.2 +	-	3.0 +	-
outParamCount()	1.2 +	-	3.0 +	-
outParameters()	1.2 +	-	3.0 +	-
resultSet()	1.2 +	-	3.0 +	-
returnValue()	1.2 +	-	3.0 +	-

Stproc.close() (Method)

Closes a stored procedure object when it is no longer required.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
JavaScript syntax:	NES	*myStproc*.close()

Although closures generally get dealt with automatically for you when a request handler exits, it is good style to call the closure methods yourself when you no longer need the database connection.

See also:	ResultSet.close()

Stproc.outParamCount() (Method)

Retrieves a count of the number of parameter values the stored procedure has returned.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	Number primitive	
JavaScript syntax:	NES	*myStproc*.outParamCount()

Calling stored procedures is not like a normal database select. Because the results of a stored procedure are not strictly the results of a simple select but have been cached and processed by the procedure, the database will not generally return them as a record but as a collection of parameters. This method tells you how many there are so you can enumerate them in your script.

See also:	ResultSet.columns()

Stproc.outParameters() (Method)

Retrieves an output parameter from the stored procedure.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`User defined`	
JavaScript syntax:	NES	*myStproc*`.outParameters(`*anIndex*`)`
Argument list:	*anIndex*	A parameter number

Given that we can obtain a count of the number of parameters returned by a stored procedure by calling the `outParamCount()` method, we can then enumerate them all and access the values that have been returned.

Stproc.prototype (Property)

The prototype for the `Stproc` object that can be used to extend the interface for all `Stproc` objects.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`Stproc` object	
JavaScript syntax:	NES	`Stproc.prototype`
	NES	*myStproc*`.constructor.prototype`

Refer to:

prototype property

Stproc.resultSet() (Method)

Retrieves a result set object from the stored procedure.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`ResultSet` object	
JavaScript syntax:	NES	*myStproc*`.resultSet()`

When you call a stored procedure in a RDMS, you don't always get back a sequence of records in the same layout and structure as when you just do a simple SQL select style query.

An SQL query would return a series of records separated by newline characters. A stored procedure might return a mixed collection of records of different types.

A `ResultSet` object is created by asking the `Stproc` object for it when the stored procedure has been called and returned from the database.

The traversing mechanisms provided with a result set allow you to move forwards through the data but you cannot move backwards. You also can only read values from a result set as opposed to a cursor which allows you to update and write new values back.

See also:	`ResultSet` object

Stproc.returnValue() (Method)

Retrieves the return value of the stored procedure.

Availability:	JavaScript – 1.2 Netscape Enterprise Server version – 3.0	
Property/method value type:	`User defined`	
JavaScript syntax:	NES	*myStproc*`.returnValue()`

This is an indication of whether the stored procedure executed successfully or not. The values returned by this may depend on the kind of database adapter your middleware employed when connecting to the database.

Stretch() (Filter/transition)

A variation on a wipe effect except that the new image appears to stretch over the old one. The old one is squashed until it disappears.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

Filter – `Stretch()`

Strictly equal to (===) (Operator/identity)

The two values must be exactly equal to one another in value and type.

Availability:	JavaScript – 1.3 JScript – 1.0 Internet Explorer – 3.02 Netscape – 4.06
Property/method value type:	`Boolean primitive`

Refer to:

Identically equal to (===)

STRIKE object (Object/HTML)

An object that represents the font style controlled by the `<STRIKE>` HTML tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*mySTRIKE* = *myDocument*.all.anElementID
	IE	*mySTRIKE* = *myDocument*.all.tags("STRIKE")[*anIndex*]
	IE	*mySTRIKE* = *myDocument*.all[*aName*]
	-	*mySTRIKE* = *myDocument*.getElementById(*anElementID*)
	-	*mySTRIKE* = *myDocument*.getElementsByName(aName)[anIndex]
	-	*mySTRIKE* = *myDocument*.getElementsByTagName("STRIKE")[*anIndex*]
HTML syntax:	`<STRIKE> ... </STRIKE>`	
Deprecated:	Yes	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

String (Type)

A native built-in type.

Availability:	ECMAScript edition – 2
Property/method value type:	`String primitive`

Entities of type `String` are collections of zero or more Unicode characters.

A string is arranged so that the characters in it can be accessed by their position. The leftmost character is considered to be at position 0 and the rightmost character is therefore at a position whose value is one less than the length of the string.

This is convenient when processing strings in a `for()` loop since you can check the enumerator against the length and as long as it is less than the length and is a positive value, it is indexing within the string.

Zero length strings are a special case.

Strings can only be accessed read-only. You cannot change the contents of a string, you can only replace it with another.

Warnings:

- You cannot reference strings with negative position values.

See also:	Cast operator, Data Type, Fundamental data type, String concatenate (+), `toString()`, Type, Type conversion, Unicode

Cross-references:

ECMA 262 edition 2 – section 4.3.17

ECMA 262 edition 2 – section 6

ECMA 262 edition 2 – section 8.4

ECMA 262 edition 3 – section 4.3.17

ECMA 262 edition 3 – section 8.4

O'Reilly *JavaScript Definitive Guide* – page 38

O'Reilly *JavaScript Definitive Guide* – page 50

String (Primitive value)

A built-in primitive value.

Availability:	ECMAScript edition – 2
Property/method value type:	`String primitive`

A string value is a member of the type String and is a finite ordered sequence of zero or more Unicode characters. There is no way to represent a single character other than by means of a very short string.

A string is not an array of characters as it would be in the C language. It is also not mutable as the other object data types that are passed by reference are. Strings are immutable and therefore to change one, you must manufacture a new string and discard the old one. This can lead to memory leaks.

Strings can contain any Unicode character code point, however, many are not available on even the most international keyboard and must be escaped. You will need to check that the host environment can render the international symbols correctly if you use them.

Strings can be delimited by either single or double quotes. This can be very useful for the occasions when a fragment of JavaScript is contained within some HTML.

Refer to the String literal topic for a list of escape characters and more information on defining string values.

Warnings:

- Beware that the HTML escaping rules come into play when JavaScript is contained within HTML quote delimited name-value pairs in tags and you must be careful to escape any characters within scripts using the JavaScript escape mechanisms and not the HTML escape mechanisms. JavaScript inside `<SCRIPT>` tags may also be affected by the host environment's escaping mechanisms.

See also:	`<SCRIPT>`, Cast operator, java.lang.String, JavaScript to Java values, String concatenate (+), String literal

Cross-references:

ECMA 262 edition 2 – section 4.3.16

ECMA 262 edition 3 – section 4.3.16

Wrox *Instant JavaScript* – page 14

String concatenate (+) (Operator/string)

Joins two string values together.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`aString1 + aString2`*
Argument list:	*`aString1`*	A string value
	`aString2`	Another string value

When the operands are a pair of strings, the plus sign will concatenate them together. This yields a single string combining both values joined end to end.

The string concatenation is not commutative. That is, the position of the two operands will affect the outcome if they are exchanged.

The addition/concatenation operator looks at the arguments and if either is a `String` already or preferentially converts to one, then a concatenation occurs. If neither argument prefers to be a `String` then a `Number` conversion happens and the values are added.

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

Warnings:

- Some conversion of type will occur if a mixture of data types is used in certain contexts.
- In string concatenations, when either of the operands is a string, the result will be a string concatenation. The same does not apply when relational expressions are involved when the subtraction operator can be used to coerce a string value into a numeric type.

See also:	Add (+), Additive operator, `Array.join()`, `Array.toString()`, Associativity, Operator Precedence, `parseFloat()`, `parseInt()`, String value, String type, String literal, `String` object, String operator, `String.split()`, `ToString`, `toString()`, Type conversion

Cross-references:

ECMA 262 edition 2 – section 11.6.1

ECMA 262 edition 3 – section 11.6.1

Wrox *Instant JavaScript* – page 37

String literal (Primitive value)

A literal constant whose type is a built-in primitive value.

Availability:	ECMAScript edition – 2
Property/method value type:	`String primitive`

A string literal is zero or more characters enclosed in matching single or double quotes. Each character may be represented by an escape sequence.

You can escape special characters with special escape sequences. You can also escape any character and specify it by its octal, hexadecimal or Unicode equivalent code point. Note that the octal values will be in the range 0 to 377 and the hexadecimal values will be in the range 0 to FF. The octal and hexadecimal escapes can only cover the first 256 character codes, some of which are control codes and should not be used anyway. The Unicode escape gives access to the full 65536 character codes in the Unicode set. Although you can specify octal or hexadecimal values, there is presently no standardized decimal-based escape mechanism. You'll just have to learn octal or hexadecimal, unfortunately.

Here are the valid common escape sequences:

Escape Sequence	Name	Symbol
`\"`	Double Quote	"
`\'`	Single Quote (Apostrophe)	'
`\\`	Backslash	\

Table continued on following page

Escape Sequence	Name	Symbol
\a	Audible alert (MSIE displays the letter a)	<BEL>
\b	Backspace (ignored silently in MSIE)	<BS>
\f	Form Feed (ignored silently in MSIE)	<FF>
\n	Line Feed (Newline – MSIE inserts a space)	<LF>
\r	Carriage Return (MSIE inserts a space)	<CR>
\t	Horizontal Tab (MSIE inserts a space)	<HT>
\v	Vertical tab (MSIE displays the letter v)	<VT>
\0nn	Octal escape	-
\042	Double Quote	"
\047	Single Quote (Apostrophe)	'
\134	Backslash	\
\xnn	Hexadecimal escape	-
\x22	Double Quote	"
\x27	Single Quote (Apostrophe)	'
\x5C	Backslash	\
\unnnn	Unicode escape	-
\u0022	Double Quote	"
\u0027	Single Quote (Apostrophe)	'
\u005C	Backslash	\
\uFFFE	A special Unicode sentinel character for flagging byte reversed text	-
\uFFFF	A special Unicode sentinel character	-

Here are some example string literals:

```
myString = "James Bond";

myString = 'Another String';

myString = 'A string with double " quotes';

myString = "He's got a single quote";
```

The characters in the quotes are converted to a String primitive value and will replace the expression in the context in which it has been used. This would normally be an assignment of a variable or perhaps part of a relational expression.

There are circumstances in HTML documents where JavaScript string delimiters may need to be single quotes, because the fragment of JavaScript is already enclosed in double quotes that are part of the HTML source code space.

For example:

```
<A HREF="javascript:passStringValue('ABCDEF');">ABCDEF</A>;
```

You can use double quotes without breaking the syntax rules of the HTML page containing the JavaScript.

You can exchange the pairs of quotes around between the contexts but it is good to stick one or the other. It tends to work out that double quotes are used in HTML which forces single quotes to be used in the JavaScript fragments that are placed into HTML tag attributes.

This is discussed with examples in the "Escaped JavaScript quotes in HTML" topic .

Warnings:

- If you use quotes in JavaScript that you plan to use inside HTML, then be sensible about the use of single and double quotes. Often you will find that double quotes will break, even though they are enclosed in single quotes when you include them in an HREF for an anchor. For example, this can break because the double quotes are seen by the HTML parser:

```
<A HREF="javascript:handleClick('bad " quotes');">
```

- Put an escape round the quotes. Sometimes a backslash is appropriate and sometimes an HTML character entity depending on who you are trying to hide the quotes from.
- Be careful if you use HTML escape sequences such as `&#39;` in JavaScript string literals. Some implementations will unescape that string in the JavaScript context and not the HTML context hoped for. That will restore the single quote inside the JavaScript string literal causing a run-time error. You should escape single quotes with a backslash.
- You cannot include a line terminator within a string literal. Instead use a newline (`\n`) escape sequence.
- Currency symbols are notoriously non-portable. Check that your target audience can display international currency symbols before using GB pounds or Euro symbols. Although these are defined in the standards, they are often missing from the installed character sets.

See also:	Escape sequence (\), Escaped JavaScript quotes in HTML, Identifier, Implicit conversion, Line terminator, Literal, String, String concatenate (+), Unicode

Cross-references:

ECMA 262 edition 2 – section 7.7.3

ECMA 262 edition 2 – section 7.7.4

ECMA 262 edition 3 – section 7.8.4

O'Reilly *JavaScript Definitive Guide* – page 38

String object (Object/core)

An object of the class "`String`".

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
JavaScript syntax:	-	*myString* = new String()
	-	*myString* = String
Object properties:	constructor, length, prototype	
Class methods:	fromCharCode()	
Object methods:	anchor(), big(), blink(), bold(), charAt(), charCodeAt(), concat(), fixed(), fontcolor(), fontsize(), fromCharCode(), indexOf(), italics(), lastIndexOf(), link(), localeCompare(), match(), replace(), search(), slice(), small(), split(), strike(), sub(), substr(), substring(), sup(), toLocaleLowerCase(), toLocaleUpperCase(), toLowerCase(), toSource(), toString(), toUpperCase(), valueOf()	

An instance of the class "`String`" is created by using the new operator on the `String()` constructor. The new object adopts the behavior of the built-in prototype object through the prototype-inheritance mechanisms.

All properties and methods of the prototype are available as if they were part of the instance.

A `String` object is a member of the type `Object`.

Cloning the built-in `String` object creates new `String` objects. This is done by calling the `String` constructor with the new operator being applied to an existing `String` object, thus:

```
myString = new String("a string of text");
```

A `String` object can be coerced to a string value and can be used anywhere where a string value would be expected.

Programmers familiar with object oriented techniques may prefer to use the `String` object while procedural language programmers may implement the same functionality with a string value instead.

This is an example of the flexibility of JavaScript in its ability to accommodate a variety of users from different backgrounds.

The prototype for the `String` prototype object is the `Object` prototype object.

In Netscape Navigator, you can traverse a string as if it were an array and access characters individually by their index position in the array. This doesn't work in MSIE.

See also:	`Native` object, `Object` object, String concatenate (+), `String.Class`, `String.length`, `String.prototype`, `unwatch()`, `watch()`

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`constructor`	1.0 +	1.0 +	2.0 +	3.02 +	-	-	2 +	-
`length`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	ReadOnly, DontDelete, DontEnum.
`prototype`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	3.0 +	2 +	-

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`anchor()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	-
`big()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`blink()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`bold()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`charAt()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`charCodeAt()`	1.2 +	5.5 +	4.0 +	5.5 +	3.0 +	3.0 +	2 +	Warning
`concat()`	1.2 +	3.0 +	4.0 +	4.0 +	-	3.0 +	3 +	-
`fixed()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`fontcolor()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`fontsize()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`fromCharCode()`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	3.0 +	3 +	-
`indexOf()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`italics()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`lastIndexOf()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`link()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	-
`localeCompare()`	1.5 +	5.5 +	6.0 +	5.5 +	-	-	3 +	-
`match()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	3 +	Warning
`replace()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	3 +	Warning
`search()`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	3 +	Warning
`slice()`	1.2 +	3.0 +	4.0 +	4.0 +	-	2.0 +	3 +	-
`small()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`split()`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	2.0 +	2 +	-
`strike()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	-
`sub()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`substr()`	1.2 +	3.0 +	4.0 +	4.0 +	-	3.0 +	3 +	-

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
`substring()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`sup()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	-	Deprecated
`toLocaleLowerCase()`	1.5 +	5.5 +	6.0 +	5.5 +	3.0 +	2.0 +	3 +	-
`toLocaleUpperCase()`	1.5 +	5.5 +	6.0 +	5.5 +	3.0 +	2.0 +	3 +	-
`toLowerCase()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`toSource()`	1.3 +	3.0 +	4.06 +	4.0 +	-	-	3 +	-
`toString()`	1.3 +	1.0 +	4.06 +	3.02 +	-	-	2 +	-
`toUpperCase()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	2.0 +	2 +	-
`valueOf()`	1.1 +	3.0 +	3.0 +	4.0 +	-	-	2 +	-

Cross-references:

ECMA 262 edition 2 – section – 4.3.18

ECMA 262 edition 2 – section – 10.1.5

ECMA 262 edition 2 – section – 15.5

ECMA 262 edition 3 – section – 4.3.18

ECMA 262 edition 3 – section – 15.5

Wrox *Instant JavaScript* – page – 33

String() constructor (Constructor)

A `String` object constructor.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0	
Property/method value type:	`String` object	
JavaScript syntax:	-	`new String()`
	-	`new String(`*`aValue`*`)`
Argument list:	*`aValue`*	Some value to be converted to a `String` object

When the `String()` constructor is used in a new expression it creates a new object based on the `String` prototype.

The value property of the new object is the same as the string that would have been returned when the constructor was invoked as a function call.

The result of this function is a String object version of the value passed in. If there is no passed-in argument an empty string `""` is returned.

Refer to the `String()` conversion function topic for a list of rules for converting other data types to strings.

Warnings:

- Note that unlike the `Object()` constructor which can be called without its parentheses, calling the `String()` constructor without them yields an uninitialized object.

See also:	Constructor function, `constructor` property, `Global` object, `new`, `Object()`

Cross-references:

ECMA 262 edition 2 – section 15.1.3.4

ECMA 262 edition 2 – section 15.5.1

ECMA 262 edition 2 – section 15.5.3.1

ECMA 262 edition 3 – section 15.5.2

String() (Function)

A `String` object constructor called as a function.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 1.0 Internet Explorer – 3.02 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`String()`
	-	`String(aValue)`
Argument list:	*aValue*	Some value to be converted to a string

When `String()` is called as a function rather than a constructor, it performs a type conversion.

The internal `ToString` conversion facilities are used for type conversion with the additional handling of a missing argument provided by the constructor itself.

Value	Result
No argument	" An empty string ".
undefined	"`undefined`".
null	"`null`".
Boolean	If the argument is true, then the result is "`true`" otherwise the result is "`false`".
Number	Special cases are provided for `NaN` and Infinity where "`NaN`" and "`Infinity`" will be returned. Otherwise the string is a textual representation of the value.
String	No conversion, the input value is returned unchanged.
Object	An internal conversion to a primitive takes place followed by a conversion from that primitive to a string. Some objects will return a string value that is immediately useful.

The result of calling this function is a string version of the value passed in. If there is no value passed in argument an empty string is returned.

Warnings:

- Converting numbers to strings can yield some strange effects due to rounding errors. Taking a numeric value and simply converting it is fairly reliable. However, the result of a numeric expression, being cast to a string directly rather than to a number variable and thence to a string, seems to expose some weaknesses in the arithmetic in some implementations.

See also:	Cast operator, Constructor function, constructor property, Implicit `conversion`, `Number()`

Cross-references:

ECMA 262 edition 2 – section – 15.1.3.4

ECMA 262 edition 2 – section – 15.5.1

ECMA 262 edition 3 – section – 15.5.1

Wrox *Instant JavaScript* – page 36

String.anchor() (Method)

Encapsulates the string within an `<A NAME="...">` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myString*.anchor()
	-	*myString*.anchor(*aName*)
Argument list:	*aName*	The value of the NAME attribute in the result string

Passing in the string "An Anchor" with the parameter "Anchor1" will result in the following string:

```
<A NAME="Anchor1">An Anchor</A>
```

This is useful for creating named anchors within the document.

See also:	Anchor object, Anchor.name, String.link(), Url object

String.big() (Method)

Encapsulates the string within an <BIG> tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Deprecated:	Yes
Property/method value type:	String primitive

JavaScript syntax:	-	*myString*.big()

Passing in the string "Big Text" will result in the following string:

```
<BIG>Big Text</BIG>
```

This is now officially deprecated in favor of the more sophisticated controls available with style sheets.

See also:	style.fontSize

String.blink() (Method)

Encapsulates the string within an `<BLINK>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Deprecated:	Yes	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myString.blink()`

Passing in the string "`Blinking text`" will result in the following string:

```
<BLINK>Blinking text</BLINK>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	`style.textDecoration`, `style.textDecorationBlink`

String.bold() (Method)

Encapsulates the string within an `<B>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Deprecated	Yes	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myString.bold()`

Passing in the string "`Bold text`" will result in the following string:

```
<B>Bold text</B>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	`style.fontWeight`

String.charAt() (Method)

A character within the string.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myString*.charAt(*aPosition*)
Argument list:	*aPosition*	A valid character position within the string

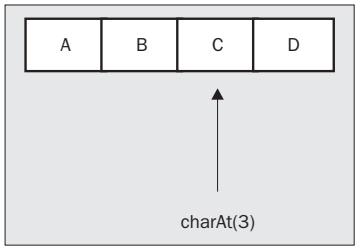
The character at the position in the string indicated by the argument value is returned by this method.

This may involve converting the receiving object to a string first. This means it can be used generically on many kinds of object.

If there is no character at that position, an empty string will be returned.

The result will be a string value and not a `String` object.

Note that there is no complementary `setCharAt()` method because strings are immutable and cannot be changed.



Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<TABLE BORDER=1>
<TR>
<TH>Index</TH>
<TH>Char</TH>
</TR>
<SCRIPT>
myString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
for(myEnum=0; myEnum<myString.length; myEnum++)
{
document.write("<TR>");
document.write("<TD>");
document.write(myEnum);
```

```
    document.write("</TD>");
    document.write(myString.charAt(myEnum));
    document.write("</TD>");
    document.write("</TR>");
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```

See also:	Character handling, Character testing, `File.stringToByte()`, `String.prototype`, `Window.atob()`

Cross-references:

ECMA 262 edition 2 – section – 15.5.4.4

ECMA 262 edition 3 – section – 15.5.4.4

String.charCodeAt() (Method)

The Unicode code point value of a character within the string.

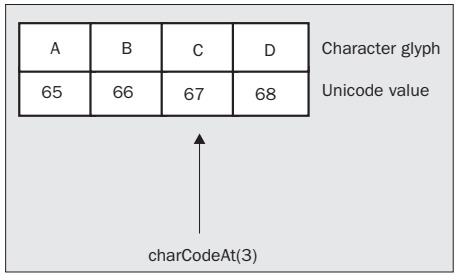
Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 5.5 Internet Explorer – 5.5 Netscape – 4.0 Netscape Enterprise Server version – 3.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myString*`.charCodeAt(`*aPosition*`)`
Argument list:	*aPosition*	A valid character position within the string

This method returns the Unicode code point of the character at the indicated position.

Note that the string index positions start at `0` and not `1`.

If there is no character at the indicated position the `NaN` value will be returned.

This method may involve the receiver being converted to a string and so it can be applied generically to many kinds of objects.



Warnings:

- If the result of calling this method creates a `null` string, then you may need to put in a special case handler. At the point where the `null` character appears, any string value will be truncated.
- For example this:

```
"AAA" + String.fromCharCode(0) + "BBB";
```

will only display the "AAA" portion of the string.

- You can create other undesirable character values as well such as the delete character at the code point 127.
- Characters at the code points 128 to 159 may display as question marks because they cannot be resolved to meaningful characters.
- Character codes 208, 222, 240 and 254 render as a caret delimited character entity description word rather than as a single character glyph. This suggests that you should not rely on the output string having the same number of characters as the input numeric array. It also suggests there may be some problems converting number arrays to binary BLOb objects using this technique.
- Note that the character code mapping does not necessarily correspond to the underlying code map of the platform the browser is running in.

Example code:

```
<HTML><HEAD></HEAD><BODY><TABLE
BORDER=1><TR><TH>Index</TH><TH>Char</TH><TH>Code</TH></TR><SCRIPT>myString =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";for(myEnum=0; myEnum<myString.length;
myEnum++){document.write("<TR>");document.write("<TD>");document.write(myEnum);doc
ument.write("</TD>");document.write("<TD>");document.write(myString.charAt(myEnum)
);document.write("</TD>");document.write("<TD>");document.write(myString.charCodeA
t(myEnum));document.write("</TD>");document.write("</TR>");}</SCRIPT></TABLE></BOD
Y></HTML>
```

See also:	Arithmetic type, Cast operator, Character handling, Character testing, Character-case mapping, `File.stringToByte()`, Integer constant, JavaScript to Java values, `String.fromCharCode()`, `String.prototype`, `Window.atob()`

Cross-references:

ECMA 262 edition 2 – section – 15.5.4.5

ECMA 262 edition 3 – section – 15.5.4.5

String.Class (Property/internal)

Internal property that returns an object class.

Availability:	ECMAScript edition – 2

This is an internal property that describes the class that a `String` object instance is a member of. The reserved words suggest that in the future this property may be externalized.

See also:	Class, `String` object

Property attributes:

`DontEnum`, `Internal`.

Cross-references:

ECMA 262 edition 2 – section – 8.6.2

ECMA 262 edition 2 – section – 15.5.2.1

ECMA 262 edition 3 – section – 8.6.2

String.concat() (Method)

A method for concatenating as opposed to the concatenate operator.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myString*`.concat(`*aString*`)`
Argument list:	*aString*	A string to be concatenated to the receiving string

The concatenation operator (+) is the more common method of concatenating strings together.

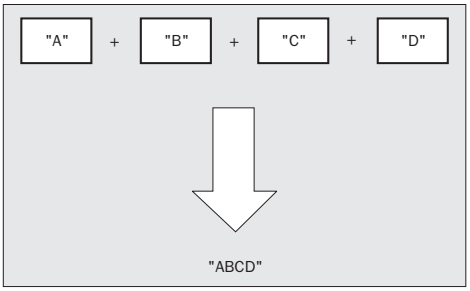
This method call:

```
myString1.concat(myString2)
```

is functionally equivalent to:

```
myString1 + myString2
```

The second form is more intuitive in its meaning, and hence is recommended in favor of the `concat()` method.



Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myString1 = "ABCDEFGHIJKLM";
myString2 = "NOPQRSTUVWXYZ";
document.write(myString1);
document.write("<BR>");
document.write(myString2);
document.write("<BR>");
document.write(myString1.concat(myString2));
document.write("<BR>");
</SCRIPT>
</BODY>
</HTML>;
```

See also:	Array.`concat()`

Cross-references:

ECMA 262 edition 3 – section – 15.5.4.6

String.constructor (Property)

A reference to a constructor object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0	
Property/method value type:	`String` object	
JavaScript syntax:	-	*myString*`.constructor`

The initial value of the `String.prototype.constructor` is the built-in `String` constructor.

You can use this as one way of creating strings although it is more popular to use the new `String()` technique.

This property is useful if you have an object that you want to clone but you don't know what sort of object it is. Simply access the constructor belonging to the object you have a reference to.

Netscape Navigator provides constructors for many objects, virtually all of them in fact, even when it is highly inappropriate to do so. MSIE is far more selective and there are some occasions when you might wish for a constructor that MSIE does not make available.

See also:	`String.fromCharCode(),String.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.5.4.1

ECMA 262 edition 3 – section – 15.5.2

String.fixed() (Method)

Encapsulates the string within an `<TT>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Deprecated:	Yes
Property/method value type:	`String primitive`
JavaScript syntax:	- *myString*`.fixed()`

Passing in the string "`ABCD`" will result in the following string:

```
<TT>ABCD</TT>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	`style.fontFamily`

String.fontcolor() (Method)

Encapsulate the string within an `<FONT COLOR="...">` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Deprecated:	Yes
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myString*.fontcolor()
	-	*myString*.fontcolor(*aColor*)
Argument list:	*aColor*	A color value to embed as an HTML tag attribute

Passing in the string "Colored text" with the parameter "RED" will result in the following string:

```
<FONT COLOR="RED">ABCD</FONT>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	FONT.color,style.color

String.fontsize() (Method)

Encapsulates the string within an <FONT SIZE="...">> tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Deprecated:	Yes	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myString*.fontsize()
	-	*myString*.fontsize(*aSize*)
Argument list:	*aSize*	A value to embed as an HTML tag attribute

Passing in the string "Small text" with the parameter "SMALL" will result in the following string:

```
<FONT SIZE="SMALL">ABCD</FONT>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	FONT.size,style.fontSize

String.fromCharCode() (Method/static)

A class-based factory method for converting numeric character codes to String objects.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server version – 3.0 Opera – 3.0

Property/method value type:	`String primitive`	
JavaScript syntax:	-	`String.fromCharCode(`*`aChar0`*`, `*`aChar1`*`, `*`aChar2`*`, ...)`
Argument list:	*`aCharN`*	a character code value

Constructs a new string from a sequence of Unicode character code point values each passed as a separate argument.

This is a static method. So called because it belongs to the `String()` constructor and not any of the string objects. This means it is analogous to a class method in other object oriented environments.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
document.write(String.fromCharCode(72, 69, 76, 76, 79));
</SCRIPT>
</BODY>
</HTML>
```

See also:	Arithmetic type, Cast operator, Character handling, Character testing, Character-case mapping, `File.byteToString()`, Java to JavaScript values, Keyboard events, `onKeyDown`, `String.charCodeAt()`, `String.constructor`, `ToUint16`, `Window.atob()`, `Window.btoa()`

Cross-references:

ECMA 262 edition 2 – section 15.5.3.2

ECMA 262 edition 3 – section 15.5.3.2

String.indexOf() (Method)

The location of a sub-string within a string.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myString`*`.indexOf(`*`aSearchString`*`)`
	-	*`myString`*`.indexOf(`*`aSearchString`*`, `*`aPosition`*`)`

Argument list:	*aPosition*	An optional valid character position within the string to indicate the search start
	aSearchString	A string to search for within the `String` object

Returns a value indicating the location of the search string within the receiving `String` object.

If the search string is not found, the value `-1` is returned.

You can indicate a starting position within the target string and if that argument is missing, the whole string will be searched. The same happens if you put the value `0` into the second parameter.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
myIndex = myString.indexOf("MN");
document.write(myString.substr(myIndex));
</SCRIPT>
</BODY>
</HTML>
```

See also:	`String.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.5.4.6

ECMA 262 edition 3 – section – 15.5.4.7

String.italics() (Method)

Encapsulates the string within an `<I>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Deprecated:	Yes	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myString*`.italics()`

Passing in the string "`Italicized text`" will result in the following string:

```
<I>ABCD</I>
```

This is now deprecated in favor of the more sophisticated controls available with stylesheets.

See also:	`style.fontStyle`

String.lastIndexOf() (Method)

The location of the rightmost sub-string within a string.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	- *myString*`.lastIndexOf(`*aSearchString*`)`	
	- *myString*`.lastIndexOf(`*aSearchString*`,` *aPosition*`)`	
Argument list:	*aPosition*	An optional valid character position within the string to indicate the search start. Missing value or 0 indicates the entire string is to be searched
	aSearchString	A string to search for within the string object

Locates the rightmost occurrence of the search string within the receiving `String` object.

If the search string cannot be found, then the value `-1` is returned.

If the starting position is not indicated, the entire string will be searched.

The result of this method is the index of the rightmost occurrence of the sub-string or `-1` if it is not found.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myString = "AAABBBCCCAAABBBCCC";
myIndex = myString.lastIndexOf("AAA");
document.write(myString.substr(myIndex));
</SCRIPT>
</BODY>
</HTML>
```

See also:	`String.prototype`

Cross-references:

ECMA 262 edition 2 – section – 15.5.4.7

ECMA 262 edition 3 – section – 15.5.4.8

String.length (Property)

The length of a `String` object's value.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myString*`.length`

The length of the `String` object's value measured in characters.

All `String` objects inherit properties and methods from the `String` prototype object. They will all return a value as a result of requesting the `length` property.

The length value of a `String` object is immutable.

This suggests that changing the value of a `String` object actually creates a new `String` object. This explains why continual string concatenations and modifications can sometimes cause a JavaScript execution to leak large amounts of memory as it runs.

See also:	`String` object, `String`.prototype

Property attributes:

`ReadOnly`, `DontDelete`, `DontEnum`.

Cross-references:

ECMA 262 edition 2 – section 15.5.5

ECMA 262 edition 3 – section 15.5.5.1

String.link() (Method)

Encapsulates the string within an `<A HREF="...">` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0

Property/method value type:	String primitive	
JavaScript syntax:	-	myString.link()
	-	*myString*.link(*aURL*)
Argument list:	*aURL*	The URL to be embedded into an HREF HTML tag attribute

Passing in the string "A link" with the parameter "Some site" will result in the following string:

```
<A HREF="AAA">ABCD</A>
```

This is useful for creating hyperlinks within the document.

See also:	LINK object, String.anchor(),Url object

String.localeCompare() (Method)

A locale sensitive string comparison.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	-	*myString*.localeCompare(*aString*)
Argument list:	*aString*	A string to compare the value of this object against

The string passed as an argument is compared with the primitive value of the receiving String object. Using simple character comparison techniques they may not match and would fail an A == B test. The locale sensitive comparison takes special international characters and locale specific text issues into account and properly matches the strings.

Cross-references:

ECMA 262 edition 3 – section – 15.5.4.9

String.match() (Method)

Searches a string using a regular expression and returns the matches in an array.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0

Property/method value type:	`Array` object	
JavaScript syntax:	-	*myObject*.match(*aPattern*)
Argument list:	*aPattern*	A regular expression pattern

This is one of the additions to the `String` object to support regular expressions.

The matches are returned in an array with each match in a separate element. You can use this to strip numeric values out of a string containing words and numbers by looking for a pattern such as:

```
/\d+/g
```

This would yield all the numeric values from a string.

The `g` attribute affects whether a single item matches or whether all of them match.

When the `g` attribute is not used, the array object has an additional property named `index`, which contains the character location where the match occurred. It also has an additional property called `input`, which contains the original string that was searched for a match.

Warnings:

- This resets the `lastIndex` property of a `RegExp` object to 0.

Example code:

```
myString = "JavaScript is good";
myLocation = myString.search(/GOOD/i);
document.write(myLocation);
```

See also:	`Array.index`, `Array.input`, `RegExp pattern`, `RegExp.exec()`, `RegExp.lastIndex`, `Regular expression`, `String.replace()`, `String.search()`, `String.split()`

Cross-references:

ECMA 262 edition 3 – section 15.5.4.10

String.prototype (Property)

The prototype for the `String` object that can be used to extend the interface for all `String` objects.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server version – 3.0 Opera – 3.0
Property/method value type:	`String` object

JavaScript syntax:	-	`String.prototype`
	-	*myString*`.constructor.prototype`

The `prototype` property refers to the built-in `String` prototype object.

The example demonstrates how to provide extensions to all instances of this class by adding a function to the prototype object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define a function that extends the capabilities of String objects
function reverse()
{
   myArray = new Array(this.length);

   for(myEnum=0; myEnum<this.length; myEnum++)
   {
      myArray[myEnum] = this.substr(myEnum,1)
   }

   myArray.reverse();

   return myArray.join("");
}
// Register the new function
String.prototype.reverse = reverse;
// Create a string object and test the String.reverse() method
myString = new String("ABCDEFGH");
document.write(myString.reverse())
document.write("<BR>")
</SCRIPT>
</BODY>
</HTML>
```

See also:	`prototype` property, `String` object, `String.charAt()`, `String.charCodeAt()`, `String.constructor`, `String.indexOf()`, `String.lastIndexOf()`, `String.length`, `String.split()`, `String.substring()`, `String.toLocaleLowerCase()`, `String.toLocaleUpperCase()`, `String.toLowerCase()`, `String.toString()`, `String.toUpperCase()`, `String.valueOf()`

Cross-references:

ECMA 262 edition 2 – section 15.2.3.1

ECMA 262 edition 2 – section 15.5.3.1

ECMA 262 edition 2 – section 15.5.4

ECMA 262 edition 3 – section 15.5.3.1

String.replace() (Method)

Searches a string using a regular expression and replace the matches.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myObject*.replace(*aPattern*, *aString*)
Argument list:	*aPattern*	A regular expression pattern
	aString	A replacement string

This is one of the additions to the `String` object to support regular expressions.

The search pattern locates matches, which are then replaced by the string value in the second argument.

The value is modified in place.

The replacement text can make use of the numbered sub-expression references and can use them in the replaced string.

Warnings:

- This resets the `lastIndex` property of a `RegExp` object to `0`.

Example code:

```
// Simple replacement
myString = "javascript, 'JAVASCRIPT', JavaScript";
myString.replace(/javascript/ig, "JavaScript");
document.write(myString);
myString.replace(/^.*\'([^\']*)\'.*$/, "---''$1''---")
```

See also:	`RegExp` pattern, `RegExp.exec()`, `RegExp.lastIndex`, Regular expression, `String.match()`, `String.search()`, `String.split()`

Cross-references:

ECMA 262 edition 3 – section – 15.5.4.11

String.search() (Method)

Searches a string using a regular expression.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myObject*.search(* *aPattern*)
Argument list:	*aPattern*	A regular expression pattern

This is one of the additions to the `String` object to support regular expressions.

The character location where the match occurred is returned. If there is no match then the value `-1` is returned instead.

If you use the global attribute ('`g`'), then it will be ignored.

Warnings:

- This resets the `lastIndex` property of a `RegExp` object to `0`.
- Don't mix this `search()` method up with the search property that belongs to the `Location` object as they are totally different things. The temptation might be to refer to the position that the string search yielded as a location. This could be very confusing for someone working on your scripts later on.

Example code:

```
myString = "JavaScript is good";
myResult = myString.search(/GOOD/i);
document.write(myResult);
```

See also:	`RegExp` pattern, `RegExp.exec()`, `RegExp.lastIndex`, Regular expression, `String.match()`, `String.replace()`, `String.split()`

Cross-references:

ECMA 262 edition 3 – section 15.5.4.12

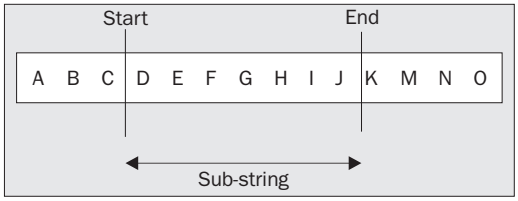
String.slice() (Method)

Returns a sub-string sliced out of the original.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server version – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myString*`.slice(`*startPos*`)`
	-	*myString*`.slice(`*startPos*`,` *endPos*`)`
Argument list:	*endPos*	The ending character position
	startPos	The starting character position

The sub-string is identified using the parameters passed as arguments in the method call.

The second argument is optional. If it is missing then it is assumed to mean the end of the string.



Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myString = "AAABBBCCCAAABBBCCC";
myIndex = myString.lastIndexOf("AAA");
document.write(myString.slice(myIndex, myIndex+4));
</SCRIPT>
</BODY>
</HTML>
```

See also:	`String.split(),String.substr(), String.substring()`

Cross-references:

ECMA 262 edition 3 – section – 15.5.4.13

String.small() (Method)

Encapsulates the string within an `<SMALL>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Deprecated:	Yes
Property/method value type:	`String primitive`
JavaScript syntax:	- *myString*.small()

Passing in the string "Small text" will result in the following string:

```
<SMALL>Small text</SMALL>
```

This is now somewhat deprecated in favor of the more sophisticated controls available with style sheets.

See also:	`style.fontSize`

String.split() (Method)

Splits a string and stores the components in an array.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`Array` object	
JavaScript syntax:	- *myString*.split(*aPattern*)	
	- *myString*.split(*aSeparator*)	
	- *myString*.split(*aSeparator*, *aCount*)	
Argument list:	*aPattern*	A regular expression to define the splitting sequence
	aSeparator	A separator string to use for slicing the string
	aCount	An iterator count to limit the number of splits

This method returns an array that contains the separated elements.

The string is split using the separator string and each separate item is stored in an array element. The items in the array are ordered in the same sequence they were presented in the original string.

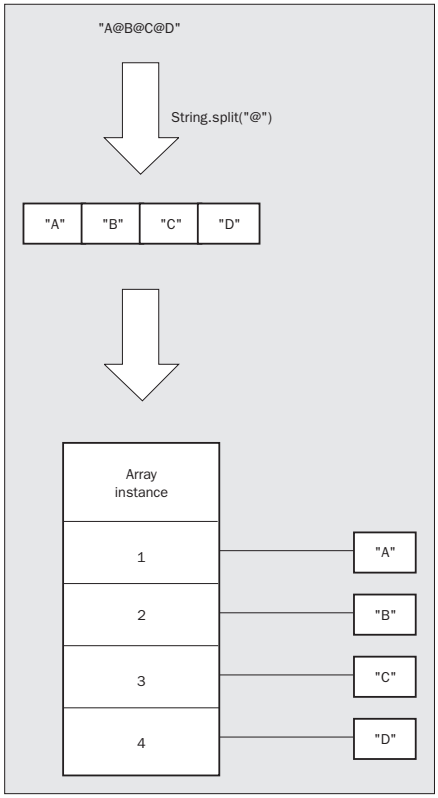
This method is the complement of the `join()` method that is applied to arrays.

The separator string is used to sub-divide the target string and is removed from the string component entities that result. The original string can be reconstructed by using the same separator in an `Array.join()` operation.

If the separator string is omitted, then the string is simply stored in the first element of a new array without being split. This is not the same as specifying an empty string. That will cause the split to happen but because you have specified an empty string, and there is an empty string between each character, the string will turned into an array of individual characters. This is amazingly useful sometimes.

In JavaScript version 1.2, this method is extended to allow the use of a regular expression as its argument. The match looks for the splitting delimiter. Using a constant character string inside the regular expression is functionally identical to using a string as the splitting value. Where this becomes powerful is where a character class or matching expression is used.

The example shows how a string of letters separated by a variety of numeric digits can be split apart using a regular expression. This would be very difficult to do any other way and would require many more lines of code.



Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myRegExp = new RegExp("[0-9]", "g");
myString = "A0B1C2D3E4F5G";
myArray = myString.split(myRegExp);
for(myEnum=0; myEnum < myArray.length; myEnum++)
{
document.write(myArray[myEnum]);
document.write("<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Array` object, `Array.join()`, `RegExp` pattern, `RegExp.exec()`, Regular expression, String concatenate (+), `String.match()`, `String.prototype`, `String.replace()`, `String.search()`, `String.slice()`, `String.substr()`, `String.substring()`

Cross-references:

ECMA 262 edition 2 – section – 15.5.4.8

ECMA 262 edition 3 – section – 15.5.4.14

String.strike() (Method)

Encapsulates the string within an `<STRIKE>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Deprecated:	Yes
Property/method value type:	`String primitive`
JavaScript syntax:	- *myString*`.strike()`

Passing in the string "Struck out" will result in the following string:

```
<STRIKE>Struck out</STRIKE>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	`style.textDecoration`, `style.textDecorationLineThrough`

String.sub() (Method)

Encapsulates the string within an `<SUB>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Deprecated:	Yes
Property/method value type:	`String primitive`
JavaScript syntax:	- `myString.sub()`

Passing in the string "Subscript text" will result in the following string:

```
<SUB>Subscript text</SUB>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	`style.fontSize`, `style.fontStyle`

String.substr() (Method)

Returns a sub-string extracted from the original.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server version – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	- `myString.substr(aStartPosition)`	
	- `myString.substr(aStartPosition, aLength)`	
Argument list:	*aStartPosition*	The index of the first character in the substring
	aLength	The length of the substring

This is a variation of the `String.substring()` method. The difference is that the substring length is passed as the second argument rather than the substring ending character index.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
myIndex = myString.indexOf("MN");
document.write(myString.substr(myIndex, 5));
</SCRIPT>
</BODY>
</HTML>
```

See also:	`String.slice(),String.split(),String.substring()`

Cross-references:

ECMA 262 edition 3 – section – B.2.3

String.substring() (Method)

Extracts a portion of a string.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myString*`.substring(`*aStartPosition*`,` *anEndPosition*`)`
Argument list:	*anEndPosition*	A location within the string to end the sub-string extraction. If this is omitted then the end of the string is assumed to be the end position.
	aStartPosition	A location within the string to begin the sub-string extraction.

If only one argument is provided, this method returns a sub-string starting at the indicated character position and proceeding to the and of the string. Where two arguments are provided the start and end points of the string are used to slice out a portion and return just that part.

When just one argument is provided, the second is supplied internally and is taken to be end of the string.

Where an argument is actually `NaN` rather than a meaningful value, then `Zero` is used instead.

If any argument is greater than the length of the string, then the length of the string is used instead.

Where the start point is larger than the end point, the two values are swapped over.

This method is intentionally generic and can be applied to non-string objects. Where necessary, the receiving object may be converted to a string so that a sub-string can be extracted.

The result returned by this method is the sequence of characters between the start and end positions of the source string. The value returned is a string primitive.

See also:	`String.prototype`, `String.slice()`, `String.split()`, `String.substr()`

Cross-references:

ECMA 262 edition 2 – section – 15.5.4.9

ECMA 262 edition 2 – section – 15.5.4.10

ECMA 262 edition 3 – section – 15.5.4.15

String.sup() (Method)

Encapsulates the string within an `<SUP>` tag context.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Deprecated:	Yes
Property/method value type:	`String primitive`
JavaScript syntax:	- *`myString`*`.sup()`

Passing in the string "`ABCD`" will result in the following string:

```
<SUP>ABCD</SUP>
```

This is now deprecated in favor of the more sophisticated controls available with style sheets.

See also:	`style.fontSize`, `style.fontStyle`

String.toLocaleLowerCase() (Method)

Converts a string to all lower case using a locale sensitive character mapping.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myString`*`.toLocaleLowerCase()`

The result of this method will be a string primitive containing the value of the object with all characters converted to lower case.

The original string is converted from arbitrary case to lower case on a character by character basis, taking account of the locale when testing characters.

The characters in the output string will be the lower case equivalents of the characters in the input string according to the rules defined by the Unicode 2.0 standard, and the locale specific rules regarding special (accented) characters.

See also:	Bit, Character handling, Character-case mapping, `String.prototype`, `String.toLocaleUpperCase()`, `String.toLowerCase()`, `String.toUpperCase()`

Cross-references:

ECMA 262 edition 3 – section 15.5.4.17

String.toLocaleUpperCase() (Method)

Converts a string to all upper case.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myString`*`.toLocaleUpperCase()`

The result of this method is a string primitive containing the value of the object with all characters converted to upper case.

The original string is converted from arbitrary case to upper case on a character by character basis, taking account of the locale when testing characters.

The characters in the output string will be the upper case equivalents of the characters in the input string according to the rules defined by the Unicode 2.0 standard, and the locale specific rules regarding special (accented) characters.

See also:	Bit, Character handling, Character-case mapping, `String.prototype`, `String.toLocaleLowerCase()`, `String.toLowerCase()`, `String.toUpperCase()`

Cross-references:

ECMA 262 edition 3 – section 15.5.4.19

String.toLowerCase() (Method)

Converts a string to all lower case.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myString`*`.toLowerCase()`

This method returns a string primitive containing the value of the object with all characters converted to lower case.

The original string is converted from arbitrary case to lower case on a character by character basis.

The characters in the output string will be the lower case equivalents of the characters in the input string according to the rules defined by the Unicode 2.0 standard.

The Canonical Unicode 2.0 mapping will be used and therefore no special international character handling semantics take place. However non-ECMA compliant implementations may provide localization support not specified by ECMA 262.

This method is intentionally generic and may be applied to a variety of object types other than strings.

The effect of this method may depend on the locale settings in a fully localizable implementation.

See also:	Bit, Character handling, Character-case mapping, `String.prototype`, `String.toLocaleLowerCase()`, `String.toLocaleUpperCase()`, `String.toUpperCase()`

Cross-references:

ECMA 262 edition 2 – section 15.5.4.11

ECMA 262 edition 3 – section 15.5.4.16

String.toSource() (Method)

Outputs a string formatted as a string literal contained in a string.

Availability:	ECMAScript edition – 3 JavaScript – 1.3 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myBoolean`*`.toString()`

This is an alternative way to deliver a string primitive version of a `String` object's value. In this case, it is formatted as a string literal and can then be used in an `eval()` function to assign another string.

If you run the example below, it should yield this as a result:

```
(new String("ABCDEF"))
```

The result of calling this method is a string version of the `String` object formatted as a string literal.

Example code:

```
// Create a number and then examine its source
myString = new String("ABCDEF");
document.write(myString.toSource());
```

String.toString() (Method)

Returns a string primitive version of an object.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 2.0 Internet Explorer – 4 Netscape – 4.06	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myString`*`.toString()`

Returns the string value as a primitive rather than an object.

For a string object, the `toString()` method returns the same thing as the `valueOf()` method.

See also:	`String.prototype`, `String.valueOf()`, `toString()`

Cross-references:

ECMA 262 edition 2 – section 15.5.4.2

ECMA 262 edition 3 – section 15.5.4.2

String.toUpperCase() (Method)

Converts a string to all upper case.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myString*`.toUpperCase()`

The result of this method is a string primitive containing the value of the object with all characters converted to upper case.

The original string is converted from arbitrary case to upper case on a character by character basis.

The characters in the output string will be the upper case equivalents of the characters in the input string according to the rules defined by the Unicode 2.0 standard.

The Canonical Unicode 2.0 mapping will be used and therefore no special international character handling semantics take place. However non-ECMA compliant implementations may provided localization support not specified by ECMA 262.

This method is intentionally generic and may be applied to a variety of object types other than strings.

The effect of this method may depend on the locale settings in a fully localizable implementation.

See also:	Bit, Character handling, Character-case mapping, `String.prototype`, `String.toLocaleLowerCase()`, `String.toLocaleUpperCase()`, `String.toLowerCase()`

Cross-references:

ECMA 262 edition 2 – section 15.5.4.12

ECMA 262 edition 3 – section 15.5.4.18

String.valueOf() (Method)

Returns the primitive value of the object.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myString*`.valueOf()`

The numeric equivalent of the string value is returned.

In the case of a `String` object, the `valueOf()` method returns the same thing as the `toString()` method.

This is proven by taking a string object and then using the == or === operators to compare the two values *myString*`.toString()` and *myString*`.valueOf()`.

See also:	Cast operator, `String.prototype`, `String.toString()`, `valueOf()`

Cross-references:

ECMA 262 edition 2 – section 15.5.4.3

ECMA 262 edition 3 – section 15.5.4.3

String operator (Definition)

Operators that act on string values.

There is only one string operator in the sense of a token that you can build a string expression out of. That is the concatenation operator. This may also be used to cast numeric values to string values by concatenating them to an empty string.

See also:	Additive operator, String concatenate (+)

Strips() (Filter/transition)

Reveals new image by sliding diagonal strips across the image.

Availability:	JScript – 5.5 Internet Explorer – 5.5

Refer to:

Filter – `Strips()`

STRONG object (Object/HTML)

An object representing the HTML content delimited by the `<STRONG>` tags.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*mySTRONG* = *myDocument*.all.*anElementID*
	IE	*mySTRONG* = *myDocument*.all.tags("STRONG")[*anIndex*]
	IE	*mySTRONG* = *myDocument*.all[*aName*]
	-	*mySTRONG* = *myDocument*.getElementById(*anElementID*)
	-	*mySTRONG* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*mySTRONG* = *myDocument*.getElementsByTagName("STRONG")[*anIndex*]
HTML syntax:	`<STRONG> ... </STRONG>`	
Argument list:	*anElementID*	The `ID` value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

`<STRONG>` tags and the objects that represent them are inline elements. Placing them into a document does not create a line break.

See also:	`Element` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onDblClick`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

STYLE object (1) (Object/HTML)

An object that encapsulates the `<STYLE>` tag in the document source as opposed to the internally created `style` objects manufactured from CSS style sheet contents.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*mySTYLE* = *myDocument*.all.*anElementID*
	IE	*mySTYLE* = *myDocument*.all.tags("STYLE")[*anIndex*]
	IE	*mySTYLE* = *myDocument*.all[*aName*]
	-	*mySTYLE* = *myDocument*.getElementById(*anElementID*)
	-	*mySTYLE* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*mySTYLE* = *myDocument*.getElementsByTagName("STYLE")[*anIndex*]
HTML syntax:	`<STYLE> ... </STYLE>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`disabled, media, readyState, type`	
Event handlers:	`onClick, onDblClick, onError, onHelp, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onReadyStateChange`	

The `<STYLE>` tag conveys no apparent visible effect on the document. It is considered to be an invisible tag.

Warnings:

- Be careful not to confuse this DOM object with the internal CSS style object that MSIE supports. The `STYLE` object is instantiated by the `<STYLE>` tag and is defined in the DOM standard. The style object is based on the CSS attributes.

See also:	`CLASS="...",Element` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
disabled	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
media	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
readyState	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
type	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5+	3.0 +	6.0 +	4.0+	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onError	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onLoad`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	3.0 +	-	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onReadyStateChange`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

STYLE.disabled (Property)

A switch to enable or disable a style object defined by the <STYLE> tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	Boolean primitive	
JavaScript syntax:	-	*mySTYLE*.disabled

If this value is set to Boolean true, the STYLE object will be inactive and will not be included in the cascaded style.

Warnings:

- This is not supported on some versions of MSIE on the Macintosh platform.

STYLE.media (Property)

A description of the target presentation media that this <STYLE> tag is applicable to. This is not widely or fully supported as yet.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*mySTYLE*.media

At some time in the future style sheet support will allow a different styling model to be applied to the document according to the context in which it is being used. At the very least one would expect screen and print support to be available. The following values are allowed for in this property (according the HTML 4.0 specification) although there is no active support for this yet in any browser:

- all
- aural
- braille
- handheld
- print
- projection
- screen
- tty
- tv

For now at least, the MSIE browser supports `all`, `print` and `screen`.

Other values can be added and you can define multiple values by constructing a comma separated list in this property. Obviously the `all` keyword should be used on its own.

One would expect the TV set-top boxes to support the `tv` media style. However, they currently don't have very good support for stylesheets at all as they are predominantly based around the HTML 3.2 standard. This can only improve as time goes by.

See also:	`style.size`

STYLE.readyState (Property)

The current status disposition of a `<STYLE>` object as it is being loaded.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`mySTYLE`*`.readyState`

This property reflects the downloading of a style sheet from a server.

Sometimes, you can design scripts to execute while the document is downloading, for example, inline scripts and at that time, you may even be able to trigger interval timed deferred executions as well.

If it is important that the document has completed loading, you can check this property for one of the following values:

State	Value
`uninitialized`	The object is first instantiated but has not begun loading.
`loading`	The object has commenced loading.
`loaded`	The object has completed loading.
`interactive`	The object is loaded but not yet closed but is ready to handle interaction.
`complete`	The object body has been closed and the loading is finished.

An object may not need to reflect the `complete` status before you can commence operating on it. Other objects may require that they are completely loaded. For example, you cannot create an `OBJECT` object that represents an `<OBJECT>` tag until the `<BODY>` has completed loading. This is because the ActiveX object construction requires a complete document body structure to attach itself to.

Every time this `readyState` value changes, it triggers an `onReadyStateChange` event call-back.

See also:	`onReadyStateChange`

Property attributes:

`ReadOnly`.

STYLE.type (Property)

The MIME type that describes the kind of style information contained in the `<STYLE>` tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*mySTYLE*`.type`

The MIME type of the style is accessible through the value of this property.

Refer to the MIME type topic for details of the available MIME types you will likely see in this property.

style object (2) (Object/CSS)

An object that represents an individual style element within a style sheet.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myStyle* `=` *myDocument*`.all.`*anElementID*`.style`
	-	*myStyle* `=` *myElement*`.style`
Argument list:	*aClassName*	The value in a `CLASS="..."` tag attribute.
	anElementName	The value in a `NAME="..."` or `ID="..."` tag attribute
	aTagName	An HTML tag name

Object properties:	azimuth, background, backgroundAttachment, backgroundColor, backgroundImage, backgroundPosition, backgroundPositionX, backgroundPositionY, backgroundRepeat, behavior, border, borderBottom, borderBottomColor, BorderBottomStyle, borderBottomWidth, borderCollapse, borderColor, borderLeft, borderLeftColor, borderLeftStyle, borderLeftWidth, borderRight, borderRightColor, borderRightStyle, borderRightWidth, borderSpacing, borderStyle, borderTop, borderTopColor, borderTopStyle, borderTopWidth, borderWidth, bottom, boxSizing, captionSide, cellSpacing, clear, clip, color, colorProfile, columnSpan, content, counterIncrement, counterReset, cssFloat, cssText, cue, cueAfter, cueBefore, cursor, direction, display, elevation, emptyCells, filter, float, floatStyle, font, fontFamily, fontSize, fontSizeAdjust, fontStretch, fontStyle, fontVariant, fontWeight, height, imeMode, important, layoutGrid, layoutGridChar, layoutGridCharSpacing, layoutGridLine, layoutGridMode, layoutGridType, left, length, letterSpacing, lineBreak, lineHeight, listStyle, listStyleImage, listStylePosition, listStyleType, margin, marginBottom, marginLeft, marginRight, marginTop, markerOffset, marks, maxHeight, maxWidth, minHeight, minWidth, orphans, outline, outlineColor, outlineStyle, outlineWidth, overflow, overflowX, overflowY, padding, paddingBottom, paddingLeft, paddingRight, paddingTop, page, pageBreakAfter, pageBreakBefore, pageBreakInside, pause, pauseAfter, pauseBefore, pitch, pitchRange, pixelBottom, pixelHeight, pixelLeft, pixelRight, pixelTop, pixelWidth, playDuring, posBottom, posHeight, position, posLeft, posRight, posTop, posWidth, quotes, renderingIntent, richness, right, rowSpan, rubyAlign, rubyOverhang, rubyPosition, scrollbar3dLightColor, scrollbarArrowColor, scrollbarBaseColor, scrollbarDarkShadowColor, scrollbarFaceColor, scrollbarHighlightColor, scrollbarShadowColor, size, speak, speakDate, speakHeader, speakNumeral, speakPunctuation, speakTime, speechRate, stress, styleFloat, tableLayout, textAlign, textAutospace, textDecoration, textDecorationBlink, textDecorationLineThrough, textDecorationNone, textDecorationOverline, textDecorationUnderline, textIndent, textJustify, textKashidaSpace, textShadow, textTransform, textUnderlinePosition, top, unicodeBidi, verticalAlign, visibility, voiceFamily, volume, whiteSpace, widows, width, wordBreak, wordSpacing, wordWrap, writingMode, zIndex, zoom
Object methods:	getAttribute(), getExpression(), item(), removeExpression(), setAttribute(), setExpression()
Event handlers:	onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp

DOM level 2 mandates that this object should really be called a `CSS2Properties` object and not a `style` object. It has become known as a `style` object due to the `Element` property name that points at it.

The `style` objects supported by MSIE and Netscape Navigator 4.x are quite different. For a start, the `style` objects are associated with each element in MSIE and are accessible quite easily via the `style` property. For Netscape Navigator 4, the style properties are documented under the `JSSTag` object. JSS style control permits the definition of element styles before an element has been instantiated into the document. You cannot use JSS to make dynamic style alterations.

With the release of Netscape 6.0, the style manipulation is virtually the same across both browsers. This works in both MSIE and Netscape 6 now:

```
document.getElementById("anID").style.backgroundColor="#003366";
```

Nearly all the attributes described in CSS level 1 and 2 are now supported by the `style` object. Because the codebase in Netscape 6.0 is totally new and it has only just been released, you may experience some minor instability in its support for styles.

When you look at the properties of the `style` object, you will observe that there are many alternative ways to define properties. This is quite commonplace where Microsoft define an interface to an object themselves or enhance one that has been defined by someone else (W3C or Netscape perhaps). For example, you can specify border attributes for all four borders but there are also separately defined attributes for each border. Although it's not that confusing, it does mean that there are a lot of additional keywords to learn and, from a parsing point of view, the more keywords there are, the slower the parser is going to be. It also means that programmers will employ a variety of different techniques which then forces the competing browser manufacturers to support the Microsoft extensions too.

Minimalist design is obviously not a priority here. Arguably there are benefits from this approach, too, and some people prefer having a variety of alternative ways to script around a problem. The `style` object could have been just as flexible with fewer properties at the expense of having scripts with a few more lines.

In the case of the border attributes, we might easily have coped with having a single border attribute that applied to all four sides because there are relatively few circumstances where we would want a different border on each side of a cell. On the other hand, we would then have lost some functionality so perhaps having individually addressable sides is beneficial, but then we could have omitted the collective reference leading to a need to explicitly define all four. At least this way, everyone's needs are catered for.

The properties for this object may apply styles to a variety of object types. Some are very specifically applicable to only a particular sub-set of objects. You should generally assume that the style attribute can apply to objects of any kind unless the property description topic enumerates a set of objects. In that case, the style attribute should only be applied to those object types and will likely be ignored by others. There is a possibility that applying a completely inappropriate styling to an object may cause problems or unpredictable behavior.

Warnings:

- Note that the object type for MSIE is a `style` object spelled without capitalization. The corresponding object type for Netscape 4 is a `JSSTag` object.
- MSIE also supports a `STYLE` object, which is instantiated by the `<STYLE>` tag. This is an object that represents an inline style element within the HTML of the document.

See also:	`CLASS="..."`, `Collection` object, `currentStyle` object, `Element` object, `Element.currentStyle`, `Element.style`, `JSSTag` object, `Layer` object, `rule.style`, `runtimeStyle` object

Property	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
`azimuth`	-	-	-	-	-	2 +	2 +	-	Warning
`background`	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	1 +	-	-
`backgroundAttachment`	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
`backgroundColor`	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	1 +	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
backgroundImage	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	1 +	-	-
backgroundPosition	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	Warning
backgroundPositionX	1.5 +	3.0 +	6.0 +	4.0 +	-	-	Proposed +	-	-
backgroundPositionY	1.5 +	3.0 +	6.0 +	4.0 +	-	-	Proposed +	-	-
backgroundRepeat	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
behavior	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	-	-	Warning
border	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderBottom	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderBottomColor	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
BorderBottomStyle	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
borderBottomWidth	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderCollapse	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	Warning
borderColor	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderLeft	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderLeftColor	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
borderLeftStyle	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
borderLeftWidth	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderRight	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderRightColor	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
borderRightStyle	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
borderRightWidth	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderSpacing	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	Warning
borderStyle	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
borderTop	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderTopColor	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
borderTopStyle	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
borderTopWidth	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
borderWidth	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
bottom	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
boxSizing	-	5.0 +	-	5.0 +	-	-	-	-	-
captionSide	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
cellSpacing	-	-	-	-	-	-	2 +	-	Warning
clear	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
clip	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	Warning
color	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	1 +	-	-
colorProfile	-	5.0 +	-	5.0 +	-	-	-	-	-
columnSpan	-	-	-	-	-	-	2 +	-	Warning
content	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
counterIncrement	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
counterReset	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
cssFloat	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
cssText	-	3.0 +	-	4.0 +	-	-	-	-	ReadOnly

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
cue	-	-	-	-	-	2 +	2 +	-	Warning
cueAfter	-	-	-	-	-	2 +	2 +	-	Warning
cueBefore	-	-	-	-	-	2 +	2 +	-	Warning
cursor	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
direction	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
display	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
elevation	-	-	-	-	-	2 +	2 +	-	Warning
emptyCells	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
filter	-	3.0 +	-	4.0 +	-	-	Proposed +	-	Warning
float	-	3.0 +	-	4.0 +	-	-	1 +	-	Warning
floatStyle	-	5.0 +	-	5.0 +	-	-	-	-	-
font	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
fontFamily	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
fontSize	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
fontSizeAdjust	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
fontStretch	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
fontStyle	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
fontVariant	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
fontWeight	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
height	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
imeMode	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
important	-	3.0 +	-	4.0 +	-	-	1 +	-	-
layoutGrid	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
layoutGridChar	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
layoutGridCharSpacing	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
layoutGridLine	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
layoutGridMode	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
layoutGridType	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
left	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	2 +	-	-
length	-	5.0 +	-	5.0 +	-	-	-	-	ReadOnly
letterSpacing	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
lineBreak	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
lineHeight	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
listStyle	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
listStyleImage	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
listStylePosition	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
listStyleType	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
margin	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
marginBottom	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
marginLeft	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
marginRight	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
marginTop	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
markerOffset	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
marks	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
maxHeight	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
maxWidth	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
minHeight	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
minWidth	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
orphans	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
outline	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
outlineColor	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
outlineStyle	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
outlineWidth	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
overflow	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	Warning
overflowX	1.5 +	5.0 +	6.0 +	5.0 +	-	-	-	-	-
overflowY	1.5 +	5.0 +	6.0 +	5.0 +	-	-	-	-	-
padding	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
paddingBottom	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
paddingLeft	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
paddingRight	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
paddingTop	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
page	1.5 +	5.5 +	6.0 +	5.5 +	-	2 +	-	-	-
pageBreakAfter	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	Warning
pageBreakBefore	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	-
pageBreakInside	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
pause	-	-	-	-	-	2 +	2 +	-	Warning
pauseAfter	-	-	-	-	-	2 +	2 +	-	Warning
pauseBefore	-	-	-	-	-	2 +	2 +	-	Warning
pitch	-	-	-	-	-	2 +	2 +	-	Warning
pitchRange	-	-	-	-	-	2 +	2 +	-	Warning
pixelBottom	-	5.0 +	-	5.0 +	-	-	-	-	-
pixelHeight	-	3.0 +	-	4.0 +	5.0 +	-	-	-	-
pixelLeft	-	3.0 +	-	4.0 +	5.0 +	-	-	-	-
pixelRight	-	5.0 +	-	5.0 +	-	-	-	-	-
pixelTop	-	3.0 +	-	4.0 +	5.0 +	-	-	-	-
pixelWidth	-	3.0 +	-	4.0 +	5.0 +	-	-	-	-
playDuring	-	-	-	-	-	2 +	2 +	-	Warning
posBottom	-	5.0 +	-	5.0 +	-	-	-	-	-
posHeight	-	3.0 +	-	4.0 +	-	-	-	-	-
position	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	2 +	-	Warning, ReadOnly
posLeft	-	3.0 +	-	4.0 +	-	-	-	-	-
posRight	-	5.0 +	-	5.0 +	-	-	-	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
posTop	-	3.0 +	-	4.0 +	-	-	-	-	-
posWidth	-	3.0 +	-	4.0 +	-	-	-	-	-
quotes	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
renderingIntent	-	5.0 +	-	5.0 +	-	-	-	-	-
richness	-	-	-	-	-	2 +	2 +	-	Warning
right	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
rowSpan	-	-	-	-	-	-	2 +	-	-
rubyAlign	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
rubyOverhang	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
rubyPosition	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
scrollbar3dLightColor	-	5.5 +	-	5.5 +	-	-	-	-	-
scrollbarArrowColor	-	5.5 +	-	5.5 +	-	-	-	-	-
scrollbarBaseColor	-	5.5 +	-	5.5 +	-	-	-	-	-
scrollbarDark ShadowColor	-	5.5 +	-	5.5 +	-	-	-	-	-
scrollbarFaceColor	-	5.5 +	-	5.5 +	-	-	-	-	-
scrollbar HighlightColor	-	5.5 +	-	5.5 +	-	-	-	-	-
scrollbarShadowColor	-	5.5 +	-	5.5 +	-	-	-	-	-
size	-	-	-	-	-	2 +	2 +	-	-
speak	-	-	-	-	-	2 +	2 +	-	Warning
speakDate	-	-	-	-	-	-	2 +	-	Warning
speakHeader	-	-	-	-	-	2 +	2 +	-	Warning
speakNumeral	-	-	-	-	-	2 +	2 +	-	Warning
speakPunctuation	-	-	-	-	-	2 +	2 +	-	Warning
speakTime	-	-	-	-	-	-	2 +	-	Warning
speechRate	-	-	-	-	-	2 +	2 +	-	Warning
stress	-	-	-	-	-	2 +	2 +	-	Warning
styleFloat	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-	-
tableLayout	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
textAlign	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
textAutospace	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
textDecoration	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
textDecorationBlink	1.5 +	3.0 +	6.0 +	4.0 +	-	-	Proposed +	-	Warning
textDecoration LineThrough	1.5 +	3.0 +	6.0 +	4.0 +	-	-	Proposed +	-	Warning
textDecorationNone	1.5 +	3.0 +	6.0 +	4.0 +	-	-	Proposed +	-	Warning
textDecorationOverline	1.5 +	3.0 +	6.0 +	4.0 +	-	-	Proposed +	-	Warning
textDecorationUnderline	1.5 +	3.0 +	6.0 +	4.0 +	-	-	Proposed +	-	Warning
textIndent	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
textJustify	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
textKashidaSpace	-	5.5 +	-	5.5 +	-	-	Proposed +	-	-
textShadow	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
textTransform	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
textUnderlinePosition	-	5.5 +	-	5.5 +	-	-	Proposed +	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
top	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	2 +	-	-
unicodeBidi	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-	-	-
verticalAlign	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	-
visibility	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	2 +	2 +	-	-
voiceFamily	-	-	-	-	-	2 +	2 +	-	Warning
volume	-	-	-	-	-	2 +	2 +	-	Warning
whiteSpace	1.5 +	5.5 +	6.0 +	5.5 +	-	2 +	1 +	-	Warning
widows	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	2 +	-	-
width	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	1 +	-	Warning
wordBreak	-	5.0 +	-	5.0 +	-	-	Proposed +	-	-
wordSpacing	1.5 +	3.0 +	6.0 +	4.01 +	-	2 +	1 +	-	Warning
wordWrap	-	5.5 +	-	5.5 +	-	-	Proposed +	-	-
writingMode	-	5.5 +	-	5.5 +	-	-	Proposed +	-	-
zIndex	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	2 +	2 +	-	Warning
zoom	-	5.5 +	-	5.5 +	-	-	-	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
getAttribute()	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-	Warning
getExpression()	-	5.0 +	-	5.0 +	-	-	-	-	-
item()	-	3.0 +	-	4.0 +	-	-	-	-	-
removeExpression()	-	5.0 +	-	5.0 +	-	-	-	-	-
setAttribute()	1.5 +	5.0 +	6.0 +	5.0 +	-	-	-	-	-
setExpression()	-	5.0 +	-	5.0 +	-	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	CSS	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	5.0 +	-	-	4.0 +	Warning

Inheritance chain:

Element object, Node object

style.azimuth (Property)

Part of the aural style control suite that defines the horizontal angle of the sound source relative to the listener.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.azimuth
CSS syntax:	azimuth: *anAngle aDirection*	
Argument list:	*anAngle*	An angle specified with a value or keyword
	aDirection	A direction specifier

The value of this property describes the angle or direction from which the sound is coming from. This assumes you have a sound system that is capable of revolving a full 360 degree panorama. A stereo system should be able to place the sound source in approximately the correct position left to right and by careful use of phasing may be able to move the sound source in and out of the sound field.

The directional value can be specified either as a value in degrees or by using one or two keywords.

An azimuthal value specified in degrees must have the `deg` suffix. Thus 90 degrees would encoded as `90deg`.

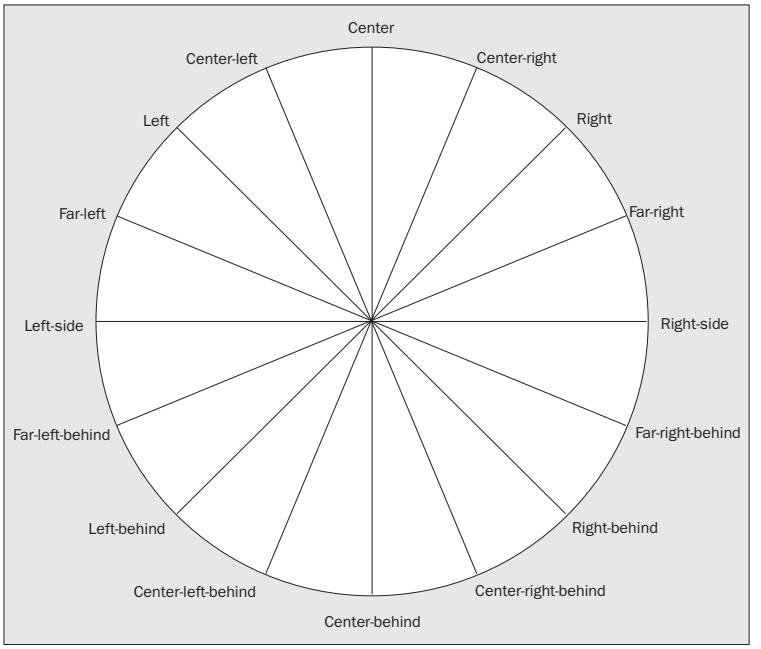
The keyword combinations and their azimuthal location are listed in the table:

Angle	Keywords
0	`center`
20	`center-right`
40	`right`
60	`far-right`
90	`right-side`
120	`far-right behind`
140	`right behind`
160	`center-right behind`
180	`center behind`
200	`center-left behind`
220	`left-behind`
240	`far-left behind`
270	`left-side`
300	`far-left`
320	`left`
340	`center-left`

In addition you can also specify these keywords:

- `leftwards`
- `rightwards`

These shift the sound source in the appropriate direction by 20 degrees.



Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, `BGSOUND.balance`, `style.elevation`

style.background (Property)

A shortcut to specify several background properties.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyle`*`.background`

CSS syntax:	`background: anAttachment aColor anImage aPosition aRepeat`	
Argument list:	*anAttachment*	Scrollability of the background
	aColor	Base color value
	anImage	Background image URL
	aPosition	Image location reference
	aRepeat	Repeat flag

This is a convenience property for specifying several properties of the background all at once. It is recommended that you specify the properties individually unless you plan to replace all of them with this property value.

The individual value are space separated from one another and are specified individually as they would be for their specific properties.

The following properties are collected into this convenience property:

- `style.backgroundAttachment`
- `style.backgroundColor`
- `style.backgroundImage`
- `style.backgroundPosition`
- `style.backgroundRepeat`

Note that the `style.backgroundPosition` value itself is composed of two values; the X coordinate followed by the Y coordinate.

If, when you define this set of properties, you omit a keyword in the namespace of any particular property, that property will be set to its default initial value.

See also:	`Background.src`, `style.backgroundAttachment`, `style.backgroundColor`, `style.backgroundImage`, `style.backgroundPosition`, `style.backgroundPositionX`, `style.backgroundPositionY`, `style.backgroundRepeat`

style.backgroundAttachment (Property)

The means of attachment for the style.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myStyle*.backgroundAttachment
CSS syntax:	background-attachment: *aSwitch*	
Argument list:	*aSwitch*	A switch setting for fixed or scrolling

This property determines how a background image, if you have one, is attached to the document on display. The following values are appropriate for this property:

- `fixed`
- `scroll`

Fixing the image allows the document content to slide over it as if the two were on separate layers.

Warnings:

- This property may be named `background-Attachment` according to some documentation.

See also:	`style.background`, `style.backgroundImage`

style.backgroundColor (Property)

A color of the styled object's background.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.backgroundColor
CSS syntax:	background-color: *acolor*	
HTML syntax:	<BODY BGCOLOR="...">	
Argument list:	*acolor*	A valid color specifier

This background color is applied to the element before any background image is drawn. You can define both a background image and a background color. If you do, and if the background image contains any pixels that are set to a transparent color, then the background color will show through.

There are several ways in which you can access the background color of a document or element. Depending on how you do this, you may affect the entire document or just a part of it.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="ONE" STYLE="background-color:RED">
The DIV block ONE
</DIV>
<DIV ID="TWO" STYLE="background-color:BLUE">
The DIV block TWO
</DIV>
<FORM>
<INPUT TYPE="button" VALUE="CLICK ME" onClick="clickMe()">
</FORM>
<SCRIPT>
//MSIE Only
function clickMe()
{
   myStyle1 = document.all.ONE.style.backgroundColor;
   myStyle2 = document.all.TWO.style.backgroundColor;

   document.all.ONE.style.backgroundColor = myStyle2;
   document.all.TWO.style.backgroundColor = myStyle1;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`color names`, `color value`, `Document.bgColor`, `JSSTag.backgroundColor`, `rgb()`, `style.background`

style.backgroundImage (Property)

The image displayed as a background to the object.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.backgroundImage
CSS syntax:	background-image: *aURL*	
HTML syntax:	<BODY BACKGROUND="...">	
Argument list:	*aURL*	A reference to an image asset

If a background image is available, then its URL is contained in this property. Changing the value in this property will replace the background with a new one, however there may be a perceptible delay while the new image is fetched from the web server.

The background image will be stationary or move with the document when it is scrolled according to the setting of the `style.backgroundAttachment` property.

See also:	`JSSTag.backgroundImage, style.background, style.backgroundAttachment`

style.backgroundPosition (Property)

The grid offset position for the background image.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.backgroundPosition`
CSS syntax:	`background-position:` *anX* `*` *aY*	
Argument list:	*anX*	A value describing the X coordinate position
	aY	A value describing the Y coordinate position

This property defines the reference position for the background image relative to the top left corner of the extent rectangle containing the styled element.

Like the `background` property, this is a convenience property which can be used instead of specifying the X and Y values separately.

This value should have both a top and a left position specified with a space character in between them. If you omit the second value, the first and only value will be assumed to apply to the horizontal axis while the value `50%` will be used by default for the vertical axis.

The positioning of the image includes the padding space around the styled element.

Oddly enough, the default measurement units for this property are percentages. You will achieve a more accurate positioning effect if you specify the value in pixels.

You can use the following keywords for vertical positioning:

- `top`
- `center`
- `bottom`

These keywords can be used for horizontal positioning:

- `left`
- `center`
- `right`

Warnings:

- Some versions of MSIE for the Macintosh do not use this value correctly and the image positioning may be unpredictable.

Example code:

```
// Example background position assignments
myStyle.backgroundPosition = "top left";
myStyle.backgroundPosition = "10 left";
myStyle.backgroundPosition = "top 20";
myStyle.backgroundPosition = "50% 50%";
myStyle.backgroundPosition = "25% 0";
```

See also:	Measurement units, `style.background`

style.backgroundPositionX (Property)

The X-coordinate of the background image grid position.

Availability:	CSS level – Proposed JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.backgroundPositionX`
CSS syntax:	`background-position-x:` *aValue*	
Argument list:	*aValue*	An X coordinate value

This property defines the horizontal coordinate of the reference position for the background image relative to the left edge of the extent rectangle containing the styled element. The positioning of the image includes the padding space around the styled element.

Unlike the `backgroundPosition` property, the default measurement units for this property are pixels. This is the most accurate means of positioning background images. You can also use the value `"left"` to equate symbolically to the value `0`.

Example code:

```
// Example background positionX assignments
myStyle.backgroundPositionX = "left";
myStyle.backgroundPositionX = "20";
myStyle.backgroundPositionX = "50%";
myStyle.backgroundPositionX = "0";
```

See also:	Measurement units, `style.background`

style.backgroundPositionY (Property)

The Y-coordinate of the background image grid position.

Availability:	CSS level – Proposed JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.backgroundPositionY`
CSS syntax:	`background-position-y:` *aValue*	
Argument list:	*aValue*	An Y-coordinate value

This property defines the vertical coordinate of the reference position for the background image relative to the top edge of the extent rectangle containing the styled element. The positioning of the image includes the padding space around the styled element.

Unlike the `backgroundPosition` property, the default measurement units for this property are pixels. This is the most accurate means of positioning background images. You can also use the value `"top"` to equate symbolically to the value `0`.

Example code:

```
// Example background positionY assignments
myStyle.backgroundPosition = "top";
myStyle.backgroundPosition = "10";
myStyle.backgroundPosition = "50%";
myStyle.backgroundPosition = "0";
```

See also:	Measurement units, `style.background`

style.backgroundRepeat (Property)

A property that controls the background step and repeat behavior.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.backgroundRepeat
CSS syntax:	background-repeat: *aValue*	
Argument list:	*aValue*	A selector for the kind of tiling algorithm

Depending on how you have designed the background image, you can use this property to control how it is step repeated across or down the page. The following values are appropriate:

- `repeat`
- `repeat-x`
- `repeat-y`
- `no-repeat`

See also:	`style.background`

style.behavior (Property)

Defines the URL location of a behavior HTC file.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.behavior

The behavior value may be specified as a file to be requested from the server or as a locally available behavior object. There are also built-in behaviors that the MSIE browser makes available.

Web-references:

http://msdn.microsoft.com/workshop/author/dhtml/reference/properties/behavior_1.asp#behavior

style.border (Property)

A border round the styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.border
CSS syntax:	border: *aWidth aStyle aColor*	
Argument list:	*aWidth*	A border width value
	aStyle	A border style selector
	aColor	A color value

This property provides a way to set all four borders in one assignment. It takes three values. However, you must specify the border style value for it to have any effect. The three values to define are:

- `borderStyle`
- `borderColor`
- `borderWidth`

Of course, you can specify them individually if you prefer, but this property may provide a cleaner display transition.

The three items should be space separated and defined as they would be for the specific properties that operate on them individually. Because the properties use distinctly different namespaces, the values can be defined in any order so long as the style value is always present.

You should include the `width` attribute. In fact unless you are specifying all three values, this shortcut isn't much of a saving over the discrete property setting mechanisms.

Borders and outlines are similar but not the same. An outline is like a border but it is drawn within the extent of the object. A border is drawn outside the object. The width of a border is measured outwards from the edge of element while the outline width is measured inwards.

See also:	color names, color value, Measurement units, `style.borderColor`, `style.borderStyle`, `style.borderWidth`, `style.margin`, `style.outline`, `style.padding`

style.borderBottom (Property)

Set the color, width and style of the bottom edge of the style border.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderBottom
CSS syntax:	`border-bottom:` *aWidth aStyle aColor*	
Argument list:	*aWidth*	A border width value
	aStyle	A border style selector
	aColor	A color value

See also:	color names, color value, Measurement units, `style.borderColor`, `style.borderStyle`, `style.borderWidth`, `style.border`

style.borderBottomColor (Property)

The color of the bottom edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderBottomColor
CSS syntax:	`border-bottom-color:` *aColor*	
Argument list:	*aColor*	A color value

See also:	color names, color value, Measurement units, `rgb()`, `style.borderColor`

style.borderBottomStyle (Property)

The type of line for the bottom edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderBottomStyle
CSS syntax:	border-bottom-style: *aStyle*	
Argument list:	*aStyle*	A border style selector

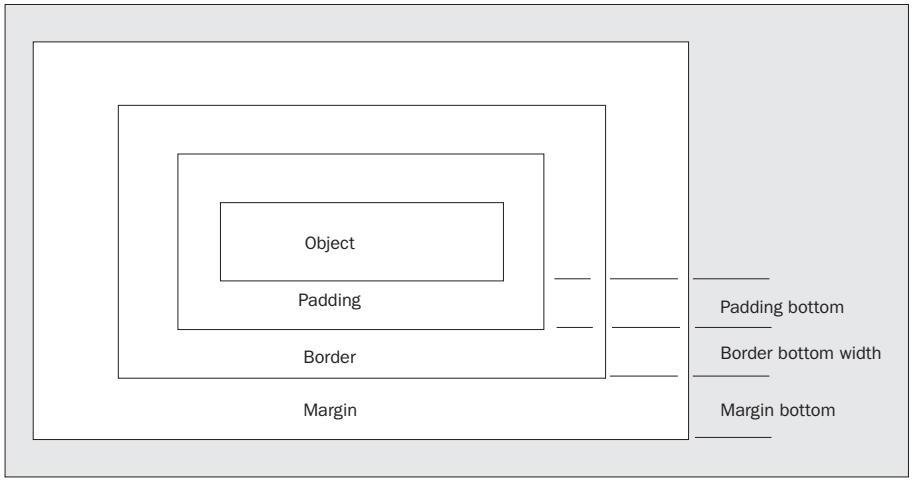
Refer to:

`style.borderStyle`

style.borderBottomWidth (Property)

The width of the bottom edge of the object's border.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderBottomWidth
CSS syntax:	border-bottom-width: *aWidth*	
Argument list:	*aWidth*	A border width value



See also:	`JSSTag.borderBottomWidth`, Measurement units, `style.borderWidth`

style.borderCollapse (Property)

A switch that determines whether borders of adjacent elements are drawn independently of one another or shared between the two items.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyle`*`.borderCollapse`
CSS syntax:	`border-collapse:` *`aSwitch`*	
Argument list:	*`aSwitch`*	A switching parameter value

Collapsing borders gives adjacent objects the same appearance as table cells. Cells in a table share common borders. The downside of this is that it is sometimes hard to see which object a border belongs to. You might turn off all the borders on an object that is surrounded by objects that have active borders. Then collapsing the borders for all the objects would look no different than if the middle object had its borders all active as well.

These keywords are appropriate for use with this property:

- `collapse`
- `separate`

Warnings:

- This style property appears to be inoperable in the Netscape 6.0 release. However it can be simulated by setting the border width to `0` for the edge that adjoins another bordered element. That should work across all CSS-compliant browsers.

See also:	`style.borderSpacing`

style.borderColor (Property)

The color of the border around an element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.borderColor`
CSS syntax:	`border-color:` *aColor*	
Argument list:	*aColor*	A color value

This property lets you change the color of one or more of the borders around an object.

If you specify only one value, all four borders are set. Two values define the top and bottom with the first and the second then applies to the left and right. If three values are specified, the first controls the top, the second controls both left and right, while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order: top, right, bottom and left.

Color values can be specified using symbolic names, `rgb()` functions, or hash delimited hex values.

This property might be used to control the border color around a frame or in fact with style sheet controls, you can control the border color around any object that can have a style associated with it.

See also:	color names, color value, `JSSTag.borderColor`, `rgb()`, `style.border`, `style.outlineColor`

style.borderLeft (Property)

Sets the color, width and style of the left edge of the style border.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderLeft
CSS syntax:	`border-left:` *aWidth* `aStyle acolor`	
Argument list:	*aWidth*	A border width value
	aStyle	A border style selector
	acolor	A color value

See also:	color names, color value, Measurement units, `style.borderColor`, `style.borderStyle`, `style.borderWidth`, `style.border`

style.borderLeftColor (Property)

The color of the left edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderLeftColor
CSS syntax:	`border-left-color:` *aColor*	
Argument list:	*aColor*	A color value

See also:	color names, color value, rgb(), style.borderColor

style.borderLeftStyle (Property)

The type of line used for the left edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.borderLeftStyle`
CSS syntax:	`border-left-style:` *aStyle*	
Argument list:	*aStyle*	A border style selector

Refer to:

`style.borderStyle`

style.borderLeftWidth (Property)

The width of the border to the left of an element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.borderLeftWidth`
CSS syntax:	`border-left-width:` *aWidth*	
Argument list:	*aWidth*	A border width value

See also:	`JSSTag.borderLeftWidth`, Measurement units, style.borderWidth

style.borderRight (Property)

Sets the color, width, and style of the right edge of the style border.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderRight
CSS syntax:	`border-right:` *aWidth aStyle acolor*	
Argument list:	*aWidth*	A border width value
	aStyle	A border style selector
	acolor	A color value

See also:	color names, color value, Measurement units, `style.borderColor`, `style.borderStyle`, `style.borderWidth`, `style.border`

style.borderRightColor (Property)

The color of the right edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderRightColor
CSS syntax:	`border-right-color:` *aColor*	
Argument list:	*aColor*	A color value

See also:	color names, color value, `rgb()`,`style.borderColor`

style.borderRightStyle (Property)

The kind of line used for the right edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderRightStyle
CSS syntax:	`border-right-style: aStyle`	
Argument list:	*aStyle*	A border style selector

Refer to:

`style.borderStyle`

style.borderRightWidth (Property)

The width of the border to the right of an element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderRightWidth
CSS syntax:	`border-right-width: aWidth`	
Argument list:	*aWidth*	A border width value

See also:	`JSSTag.borderRightWidth`, Measurement units, `style.borderWidth`

style.borderSpacing (Property)

This defines the spacing between the adjacent bordered edges of the two elements.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderSpacing
CSS syntax:	`border-spacing:` *aValue*	
Argument list:	*aValue*	A border spacing value

If borders are not collapsed and hence elements have borders that are independent of one another, this defines the spacing between the adjacent bordered edges of the two elements.

You can space elements apart, taking into account the fact that they have borders; this interposes some space between the objects so that you can clearly see the border around each one.

Warnings:

- Because the `borderCollapse` style property is not yet working correctly in Netscape 6.0, this one may also not work properly in all situations.

See also:	`style.borderCollapse`

style.borderStyle (Property)

The style of border that is drawn round a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderStyle
CSS syntax:	`border-style:` *aStyle*	
Argument list:	*aStyle*	A border style selector

This property lets you change the style of one or more of the borders around an object.

If you specify only one value, all four borders are set. Two values define the top and bottom with the first, and the second then applies to the left and right. If three values are specified, the first controls the top, the second controls both left and right, while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order: top, right, bottom and left.

Style values can be specified using symbolic names as follows:

- `solid`
- `dashed`
- `dotted`
- `double`
- `inset`
- `outset`
- `groove`
- `ridge`
- `hidden`
- `none`

This property might be used to control the border style around a frame, or in fact with stylesheet controls you can control the border style around any object that can have a style associated with it.

Warnings:

- There may be some differences in the way that browsers draw these border styles. In particular, MSIE version 4 on Windows may not draw the dashed and dotted borders correctly.

See also:	`JSSTag.borderStyle`, `style.border`, `style.outlineStyle`

style.borderTop (Property)

Set the color, width and style of the top edge of the style border.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderTop
CSS syntax:	border-top: *aWidth aStyle aColor*	

Argument list:	*aWidth*	A border width value
	aStyle	A border style selector
	acolor	A color value

See also:	color names, color value, Measurement units, `style.borderColor`, `style.borderStyle`, `style.borderWidth`, `style.border`

style.borderTopColor (Property)

The color of the top edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderTopColor
CSS syntax:	`border-top-color:` *aColor*	
Argument list:	*aColor*	A color value

See also:	color names, color value, `rgb()`, `style.borderColor`

style.borderTopStyle (Property)

The kind of line used for the top edge of the style border.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.borderTopStyle
CSS syntax:	`border-top-style:` *aStyle*	
Argument list:	*aStyle*	A border style selector

Refer to:

`style.borderStyle`

style.borderTopWidth (Property)

The width of the border along the top edge of an element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.borderTopWidth`
CSS syntax:	`border-top-width:` *aWidth*	
Argument list:	*aWidth*	A border width value

See also:	`JSSTag.borderTopWidth`, Measurement units, `style.borderWidth`

style.borderWidth (Property)

The thickness of the style border.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.borderWidth`
CSS syntax:	`border-width:` *aWidth*	
Argument list:	*aWidth*	A border width value

This property lets you change the width of one or more of the borders around an object.

If you specify only one value, all four borders are set. Two values define the top and bottom with the first, and the second then applies to the left and right. If three values are specified, the first controls the top, the second controls both left and right, while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order: top, right, bottom and left.

Width values can be specified using symbolic names or pixel values. The following symbolic names are available:

- `thin`
- `medium`
- `thick`

This property might be used to control the border width around a frame or in fact, with style sheet controls you can control the border width around any object that can have a style associated with it.

See also:	`JSSTag.borderWidths()`, Measurement units, `style.border`, `style.margin`, `style.outlineWidth`

style.bottom (Property)

A positioning reference point.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyle`*`.bottom`
CSS syntax:	`bottom: `*`aValue`*	
Argument list:	*`aValue`*	A positioning value

A CSS-P positioning style attribute that controls the location of an element relative to its containing parent element. The bottom edges of the two elements are used as the reference points.

The value can be specified in the usual pixel or fractional em-dash measurement units or the `auto` keyword can be used to let the browser do the positioning itself.

The exact positioning is affected by settings for `padding`, `border`, `margin`, and (particularly the mode of) the `position` property.

See also:	Measurement units, `style.left`, `style.pixelBottom`, `style.posBottom`, `style.right`, `style.top`

style.boxSizing (Property)

A special style supported by MSIE to control the way that elements are boxed.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.boxSizing
CSS syntax:	box-sizing: *aControl*	
Argument list:	*aControl*	The box sizing control value

Although the property is present as an enumerable property, searching the Microsoft documentation base yielded no useful information. Some very sparse web search results suggested this was some kind of layout control facility.

style.captionSide (Property)

An attribute that controls the positioning of a caption with respect to its owning object.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.captionSide
CSS syntax:	caption-side: *aSwitch*	
Argument list:	*aSwitch*	A switching value

This property would normally be used on a style that is used with tables. It allows the caption position relative to the owning element to be controlled.

The following keywords can be used with this property:

- `top`
- `bottom`

This is intended to replace the deprecated alignment controls that were formerly available with the `<CAPTION>` HTML tag.

See also:	`CAPTION` object, `CAPTION.align`, `CAPTION.vAlign`, `TABLE` object

style.cellSpacing (Property)

Defines the cell spacing of items in a table.

Availability:	CSS level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.cellSpacing
CSS syntax:	`cell-spacing: aValue`	
Argument list:	*aValue*	A spacing value

This property is intended to provide control over the cell spacing for tables. The property is able to accept one or two length values specified in the normal measurement units.

If you specify just one value it is used for both horizontal and vertical cell spacing. If you specify both values, the horizontal is taken to be first and the vertical spacing uses the second.

Assigning the `none` keyword to the property resets the cell spacing to the browser default value.

Warnings:

- This CSS 2 property is not yet supported by MSIE.

See also:	Measurement units, `TABLE` object

style.clear (Property)

A means of controlling text flow and positioning of objects adjacent to one another to allow them to coexist on the same horizontal line or to force a line break before or after them.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.clear
CSS syntax:	`clear: aValue`	
Argument list:	*aValue*	An alignment control value

This property is useful for giving fine control over the text flow around images or other objects. You can use it to clear alignment settings either side of the object, making it behave like a block structured item as far as the text flow is concerned. The following values can be assigned to this property:

- `none`
- `left`
- `right`
- `both`

The exact behavior depends on the presence of adjacent elements and the settings of their `style.float` properties being defined as either `left` or `right`.

See also:	`JSSTag.clear`, `style.float`

style.clip (Property)

A clip region for a style.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.clip`
CSS syntax:	`clip:` *aValue*	
Argument list:	*aValue*	Either a rectangle object or a keyword value

Positionable objects can have a clipping region defined for them. This defines the area of the styled object that is visible. At present you can only clip to a rectangular shape. You can specify either a clipping region measured in pixels, or use the symbolic name "`auto`" to use the extent rectangle surrounding the object as its clip region. The coordinates should be specified in the order: top, right, bottom and left and need to be enclosed in a `rect()` constructor function to instantiate a `rect` object to be assigned to the property.

You should be careful not to exceed the extent rectangle of the containing parent element. This should not cause a problem and the clipping regions should be clipped one within the other, but MSIE does not handle this very gracefully sometimes.

If you access the `left`, `right`, `top` and `bottom` properties of the `style.clip` `rect` object individually, you can specify a single numeric value for each as need be.

Warnings:

- This does not work in all versions of MSIE for Macintosh.

See also:	`Clip` object, `Rect` object

style.clip.bottom (Property)

The bottom edge of an element object's clip region.

Availability:	CSS level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.clip.bottom`

Refer to:

`style.clip`

style.clip.left (Property)

The left edge of an element object's clip region.

Availability:	CSS level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.clip.left`

Refer to:

`style.clip`

style.clip.right (Property)

The right edge of an element object's clip region.

Availability:	CSS level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.clip.right`

Refer to:

`style.clip`

style.clip.top (Property)

The top edge of an element object's clip region.

Availability:	CSS level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.clip.top`

Refer to:

`style.clip`

style.color (Property)

Defines the foreground color of any text drawn in the content of the styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0

Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.color
CSS syntax:	color: *aColor*	
Argument list:	*aColor*	A valid color value

The color of foreground text affected by this style object will be defined in this property.

The color can be specified in the normal way according to the HTML color specifiers.

See also:	color names, color value, `JSSTag.color, rgb(), String.fontcolor(),style.renderingIntent`

style.colorProfile (Property)

This is an extension to the style model that allows for accurate color models to be used for improved color fidelity of the displayed image.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.colorProfile
CSS syntax:	color-profile: *aProfile*	
Argument list:	*aProfile*	A color profile selector

This is related to the ICC color profiling support in the browser and platform. See also the Apple ColorSync technologies.

See also:	`style.renderingIntent`

style.columnSpan (Property)

Defines the number of columns to span when displaying a table cell.

Availability:	CSS level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.columnSpan

CSS syntax:	column-span: *aCount*	
Argument list:	*aCount*	A number of columns to span

The columnSpan property is used to define how many columns a table cell should span. There are other alternative ways to access the cell spanning controls for a table cell via the DOM representation of tables through the TABLE, TD and TH objects.

You should specify a numeric value to indicate how many columns the table cell should span. This can also be used with the COL and COLGROUP objects.

Warnings:

- This style property is not yet supported by MSIE.

See also:	COL object, COLGROUP object, TABLE object, TABLE.cols, TD.colSpan, TH.colSpan

style.content (Property)

A means of adding small fragments of HTML before and after an element without that HTML needing to be coded into the document source with the styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.content
CSS syntax:	content: *someContentRef*	
Argument list:	*someContentRef*	A URL, string or value to be placed before or after an element

If you have a generic effect you want to achieve that goes beyond what is possible with a simple property or style attribute and can only be accomplished by additional fragments of HTML, this property provides a way to add fragments of HTML before and after an element.

The string to be assigned to this should be a valid fragment of CSS text that uses the before: and after: pseudo elements to define leading and trailing fragments of HTML.

This is a relatively new part of the CSS standard and does not yet have a well defined JavaScript binding.

style.counterIncrement (Property)

A means of controlling the way that counters in enumerated items are to be incremented when they are used in the document.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.counterIncrement
CSS syntax:	counter-increment: *aValue*	
Argument list:	*aValue*	An incrementing value for the counter

The value specified in this property is added to the counter as it is enumerated in an ordered list.

See also:	style.listStyle,style.listStyleType

style.counterReset (Property)

A way to reset a counter at the start of a section.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.counterReset
CSS syntax:	counter-reset: *aValue*	
Argument list:	*aValue*	A new initial value for the counter

If you are building complex ordered list structures, you may want to force a counter to be reset to a specific value. This property provides a means of assigning a new value to the list at the start of the enumeration loop.

style.cssFloat (Property)

An extension to the `float` attribute that is a standard CSS attribute. `cssFloat` is an MSIE-only attribute and is not standardized. It appears to simply be another name for the `float` property.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.cssFloat
CSS syntax:	`css-float: anAlignment`	
Argument list:	*anAlignment*	A float control word

See also:	`style.styleFloat,style.float`

style.cssText (Property)

The CSS style sheet specification source text for this style object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.cssText

This property returns the contents of the entire style sheet rule that this style object belongs to. You can also assign a new CSS text value to this property in accordance with the style sheet syntax rules.

Property attributes:

`ReadOnly`.

style.cue (Property)

Part of the aural style control suite that defines the aural icon sound before and after an item is spoken.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.cue
CSS syntax:	cue: *aBefore anAfter*	
Argument list:	*aBefore*	A cue sound before the speech
	anAfter	A cue sound after the speech

This property is part of the audible style sheet property set. It is used as a shortcut for defining the value of the `style.cueBefore` and `style.cueAfter` properties in a single assignment.

Warnings:

- This style property is not yet supported by any browsers.

See also:	Aural style sheets, `style.cueAfter`, `style.cueBefore`

style.cueAfter (Property)

Part of the aural style control suite that defines the aural icon sound after an item is spoken.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.cueAfter
CSS syntax:	cue-after: *aSound*	
Argument list:	*aSound*	A sound to be played after speech

The `cueAfter` property provides a way to define a short audible sound after a spoken phrase. You can specify the URI for an audio clip or define the value `none` to inhibit the audible cue following the phrase.

The URI may be relative to the document or a fully specified URL value. It should locate an audio file having a MIME type that the browser supports for playback.

Warnings:

- This style property is not yet supported by any browsers.

See also:	Aural style sheets, MIME types, `style.cue`

style.cueBefore (Property)

Part of the aural style control suite that defines the aural icon sound before an item is spoken.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*`.cueBefore`
CSS syntax:	`cue-before:` *aSound*	
Argument list:	*aSound*	A sound to be played after speech

The `cueBefore` property provides a way to define a short audible sound before a spoken phrase. You can specify the URI for an audio clip or define the value `none` to inhibit the audible cue preceding the phrase.

The URI may be relative to the document or a fully specified URL value. It should locate an audio file having a MIME type that the browser supports for playback.

Warnings:

- This style property is not yet supported by any browsers.

See also:	Aural style sheets, MIME types, `style.cue`

style.cursor (Property)

A cursor shape to display when the pointer hovers over the element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myStyle*.cursor
CSS syntax:	cursor: *aShape*	
Argument list:	*aShape*	One of the available cursor shapes

You can define the shape of the cursor when the mouse pointer moves over the element. This is far more sensible than defining a onMouseOver and onMouseOut handler to set and reset the cursor shape.

The following cursor names can be specified:

- default
- auto
- crosshair
- help
- move
- pointer
- text
- wait
- hand
- resize
- n-resize
- ne-resize
- e-resize
- se-resize
- s-resize
- sw-resize
- w-resize
- nw-resize

The CSS standard suggests that a URI value could be specified to allow for a downloadable cursor shape to be defined although this is not currently supported in any browser.

style.direction (Property)

Controls the direction of flow of inline content such as text and table cells.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.direction
CSS syntax:	direction: *aDirection*	
Argument list:	*aDirection*	One of the available text directions

The direction property may be set to indicate a left to right or right to left parsing direction.

This is part of the localization support and represents the contents of the `DIR="..."` tag attribute.

If you assign a value to this property it is case-sensitive and must be one of the following:

- `ltr`
- `rtl`
- `ltr-override`
- `rtl-override`

This property works in conjunction with the `lang` property to control the direction of text flow.

The variations with the `override` modifier keyword are intended to force the direction even when the underlying language does not support it natively according to the Unicode rules.

style.display (Property)

A control attribute that defines how an element should be rendered into the display window.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myStyle*.display
CSS syntax:	display: *aType*	
Argument list:	*aType*	One of the supported display types

This is currently implemented as a simple visibility switch. If the property is set to the value "none", then the object is hidden. To reveal the object again, set this property to its default value which is an empty string ("").

When the object is hidden, the surrounding objects close up the space. This means you cannot use the property to accomplish a blinking effect, because everything will dance around the screen as the object appears and disappears.

The full CSS specification for this property allows for it to control the display of an object as either an inline or block level element in the page.

The following keywords are defined for use with this property:

- none
- block
- compact
- inline
- inline-table
- list-item
- run-in
- table
- table-caption
- table-cell
- table-column-group
- table-footer-group
- table-header-group
- table-row
- table-row-group

See also:	JSSTag.display

style.elevation (Property)

Part of the aural style control suite that defines the height of a sound source within a 3D space.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*`myStyle`*`.elevation`
CSS syntax:	`elevation: `*`aValue`*	
Argument list:	*`aValue`*	One of the available elevation values

The sound source can be located in the vertical direction giving a full 3D spatial system around the surface of a sphere when combined with the `azimuth` property.

The following values are appropriate:

value	elevation
`level`	On the same horizontal plane as the listener
`above`	Directly overhead
`below`	Directly underneath
`higher`	Approximately 10 degrees higher than the previous value
`lower`	Approximately 10 degrees lower than the previous value

Values can be specified in degrees with the `deg` suffix. The values `+90deg` and `-90deg` correspond to the `above` and `below` keywords.

Warnings:

- This style property is not yet supported by any browsers.

See also:	Aural style sheets, `style.azimuth`

style.emptyCells (Property)

An indication of how empty cells in a table should be displayed.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.emptyCells
CSS syntax:	empty-cells: *aControl*	
Argument list:	*aControl*	What to do with empty cells in a table

This property controls how the empty cells in a table are displayed. It can accommodate the following values:

- show
- hide

Arguably it might be useful to be able to specify some alternatives to simply hiding or showing the cells. Refer to the URL for an informative discussion on empty cells in tables.

See also:	TABLE object

Web-references:

http://www.hut.fi/u/jkorpela/HTML/emptycells.html

style.filter (Property)

Defines the visual, reveal or blend filter for the object.

Availability:	CSS level – Proposed JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myStyle*.filter
CSS syntax:	filter: *aFilter*(*aParam*=*aValue*, ...) ...	
Argument list:	*aFilter*	A filter name
	aParam	A filter parameter name
	aValue	A filter parameter value

This property defines a visual effect that is used when the display is updated as the result of a change to the content of an element.

There are three kinds of filters that can be applied to an object.

- `Visual`
- `Reveal`
- `Blend`

A visual filter is used to enhance the visual appearance of objects. For example, to flip them over, add a glow effect or a drop shadow.

A reveal filter is used to apply a transition effect as the appearance changes.

A blend filter controls the speed at which a reveal filter is applied.

You can define more than one filter, they just need to be space-separated from one another.

Here is a list of the procedural filter function names:

- `AlphaImageLoader()`
- `Gradient()`

Here is a list of the static filters supported at version 5.5 of the MSIE browser:

- `Alpha()`
- `BasicImage()`
- `Blur()`
- `Chroma()`
- `Compositor()`
- `DropShadow()`
- `Emboss()`
- `Engrave()`
- `Glow()`
- `Light()`
- `MaskFilter()`
- `Matrix()`
- `MotionBlur()`
- `Pixelate()`
- `Shadow()`
- `Wave()`

The old `blendTrans()` and `revealTrans()` filters are now replaced by these transition filters:

- `Barn()`
- `Blinds()`
- `CheckerBoard()`
- `Fade()`
- `GradientWipe()`
- `Inset()`
- `Iris()`
- `Pixelate()`
- `RadialWipe()`
- `RandomBars()`
- `RandomDissolve()`
- `Slide()`
- `Spiral()`
- `Stretch()`
- `Strips()`
- `Wheel()`
- `Zigzag()`

Filters are defined as if they were a sequence of space delimited function calls. They aren't really functions because their argument passing mechanism is not truly JavaScript based. Arguments to each filter function are defined as `name=value` pairs. These correspond to the properties and method invocations of the underlying filter object.

Refer to the specific topics on each filter function for details of what it does and how you can control it.

When using the filters in the context of the `Style` object, the function name for each filter must be preceded by this string:

"`progid:DXImageTransform.Microsoft.`"

You can apply the filters directly as properties of the `filter` object that belongs to HTML element objects themselves.

Visual filters require that the target objects have enough layout information to enable the filter to work. This means they require height and width to be defined using absolute positioning or setting the `contentEditable` property flag to `true`.

Warnings:

- Filters are not supported in all versions of MSIE on the Macintosh. In fact, they are not really well supported outside of the MSIE browser on the Win32 platform.
- There are various sources of documentation about these filters. There is some difference between them regarding the spelling of the filters' names and the availability of the filters. The naming conventions are sometimes all lower case and at others a mixed upper and lower case. This suggests that the filter name parser may be case-insensitive. This also applies to the `name=value` pairs that are passed as arguments to the filter functions.
- Certain filter functions are no longer included in the MSDN reference material and so they may be considered to be deprecated.
- We have conformed to the case style of the MSDN reference and have included all the filters that were encountered in our source material. Those that appear not to be in the current MSDN reference are marked as deprecated as follows:

```
FlipH()
FlipV()
Grayscale()
Invert()
Mask()
XRay()
```

- These are deprecated filters that used to provide blends and reveals:

```
BlendTrans()
RevealTrans()
```

- Note that the functionality and availability of the filters has changed significantly from version 4.0 to version 5.5 of the MSIE browser.
- The old functionality has not been lost. Instead, it has been reorganized and factored into the new filter suite. Nothing that you could have done before has been taken away but you will have to address the filters differently.

See also:	`Element.filters[]`, `onFilterChange`, Procedural surfaces, Static filters, `style.textShadow`, Transition, Visual filters

style.float (Property)

An alignment control that indicates how text is to be flowed round the object that the style is applied to.

Availability:	CSS level – 1 JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`String primitive`

JavaScript syntax:	IE	*myStyle*.float
CSS syntax:	float: *anAlignment*	
Argument list:	*anAlignment*	A float control word

This property defines which side is used for the reference alignment so that text can flow round the other.

Setting the property value to `none` allows the object to be placed inline according to where it appears in the document source.

The following keywords can be used with this property:

- `left`
- `right`
- `none`

Warnings:

- `IMG` elements in Netscape Navigator 4 do not align in the same way as MSIE browser `IMG` elements. Since you can't access the style model in the same way, this is maybe a moot point but there are alternative ways to control alignment in each browser. So, assuming you could define the float attribute in both, Netscape Navigator 4 would still not position the `IMG` objects in the same way.
- The JavaScript language reserves the `float` keyword and so it is dangerous to use it as a property on any object. Because of this, a more consistent control of alignment for `IMG` objects can be achieved with the `align` property instead.
- Netscape 6.0 supports the `cssFloat` property which is preferred and which avoids the conflict with the `float` keyword.

See also:	style.clear, style.styleFloat

style.floatStyle (Property)

An MSIE extension to the normal float style attributes. All the indications are that this is simply another name for the `styleFloat` property.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myStyle*.floatStyle
CSS syntax:	float-style: *anAlignment*	
Argument list:	*anAlignment*	A floating alignment control value

See also:	style.styleFloat

Refer to:

style.styleFloat

style.font (Property)

A special shortcut styling control that provides a way to define several font styling attributes in a single assignment.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.font
CSS syntax:	font: *aStyle aVariant aWeight aSize aLineHeight aFamily*font: *aConstant*	
Argument list:	*aStyle*	A font style value
	aVariant	A font variant
	aWeight	A font weight
	aSize	A font size
	aLineHeight	A line height value
	aFamily	A font family
	aConstant	A CSS font constant

This property can be assigned with a string containing space-separated keywords and values. The values are then unpacked and assigned to individual font styling properties.

The following font styling properties are collected together into this item:

- style.fontFamily
- style.fontSize
- style.fontStyle
- style.fontVariant
- style.fontWeight
- style.lineHeight

Refer to the individual topics for these properties for details of the range of suitable values. For the style.font property, the values can be presented in any order since their namespaces do not collide.

As an alternative the CSS font constant values can be used to define font appearance according to a browser and platform specific macro. The following macros are supported:

- `caption`
- `icon`
- `menu`
- `messagebox`
- `smallcaption`
- `statusbar`

These appearance of fonts styled according to these values are not defined in any standard but they will be consistent with text presented in the appropriate contexts on the client platform.

See also:	`style.fontFamily`, `style.fontSize`, `style.fontStyle`, `style.fontVariant`, `style.fontWeight`, `style.lineHeight`

style.fontFamily (Property)

A list of fonts to be used for the element. The first one in the list that is available will be used.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.fontFamily
CSS syntax:	font-family: *aFamily* ...	
Argument list:	*aFamily*	A font family list

The ordering of the font family names dictates the priority with which they are used. The names should be separated from one another by spaces but some font names may have spaces in them, so if you use font names containing spaces, they should be enclosed inside quotes. Make sure that you use a different kind of string delimiter quote for indicating font names to that which you use to enclose the string of font names.

As well as the names of any font families that you think may be installed in the target client browser, you can specify generic font families with the following keywords:

- `serif`
- `sans-serif`
- `cursive`
- `fantasy`
- `monospace`

See also:	JSSTag.fontFamily, String.fixed(), style.font, style.voiceFamily

style.fontSize (Property)

Controls the size of the text drawn with the current font. Note that different browsers support a text-imaging model at varying resolutions, and it is difficult to obtain consistent results when older browsers are used.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.fontSize
CSS syntax:	font-size: *aSize*	
Argument list:	*aSize*	A font size

Font sizes can be specified in a variety of measurement units.

There are some issues to do with font sizes that make fonts very difficult to control across platforms. This is affected by several factors:

- Font digitizing
- Font design (x-height)
- Bitmap vs. Adobe vs. TrueType
- Pixel DPI settings
- Different defaults for symbolic sizes

There are differences between the way that fonts are rendered. Considering that the Macintosh supports at least three simultaneous font rendering models at a time and historically there have been several radically different font management schemes, it is amazing that fonts ever appear in the way they were intended. The Windows platform must provide at least half as many again while the X-Windows support on the Linux platform supports a completely different font rendering model.

Certain font families appear to render larger than others for the same point size. This difference is due to the different x-height for the characters.

The differences between the Macintosh and the Windows environment are due to the 72 DPI vs. 96 DPI default pixel resolution of the screen displays. The latest versions of MSIE for the Macintosh provide a 96 DPI switch to work round this.

The symbolic names for font sizes do not render identically across platforms nor do they render the same across browsers within the same platform. These are the symbolic font size names for absolute sizes:

- `xx-small`
- `x-small`
- `small`
- `medium`
- `large`
- `x-large`
- `xx-large`

You can use numeric values with suffixes to indicate the units of measure:

- `px`
- `em`
- `percentage`

The `px` suffix means pixels while the `em` value means the width of an em-dash in the current font. You can use floating point values for measurements in `em` units. You can also use floating point values for percentage values. The percentage relates to the containing parent object's font settings.

Two additional symbolic names are reserved for relative font sizes:

- `larger`
- `smaller`

The larger and smaller keywords index up and down the scale defined by the absolute size keywords.

Warnings:

- Be very careful when defining styles with relative settings. This can cascade recursively in some cases and you will end up with text that reduces or increases in size as it goes down the page.

See also:	`JSSTag.fontSize,String.big(),String.fontsize(), String.small(),String.sub(), String.sup(), style.font`

style.fontSizeAdjust (Property)

A means of compensating for the differences in browser font rendering models.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.fontSizeAdjust
CSS syntax:	font-size-adjust: *aValue*	
Argument list:	*aValue*	An adjustment factor

This is a mechanism that is intended to help with adjustments for the x-height differences between fonts. It provides a compensation effect that reduces fonts with large x-heights so that they appear to be the same size as other fonts with small x-heights.

The following values can be assigned to this property:

- z
- none

style.fontStretch (Property)

A means of extending the font in the horizontal direction only.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.fontStretch
CSS syntax:	font-stretch: *aValue*	
Argument list:	*aValue*	A font stretch control value

This styling property is used to change the horizontal span of the text by expanding or contracting it. The property can accept the following keywords:

- normal
- wider
- narrower
- ultra-condensed
- extra-condensed
- condensed
- semi-condensed
- semi-expanded
- expanded
- extra-expanded
- ultra-expanded
- inherit

style.fontStyle (Property)

Controls the italicization of a font. The oblique and italic styles affect the displayed font in different ways.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.fontStyle
CSS syntax:	font-style: *aStyle*	
Argument list:	*aStyle*	A font style value

Fonts can be styled to present themselves in a slanted appearance. It is not commonly realized that an italic and oblique font are not the same. A fully featured font engine provides both oblique and italic font styles. A less well featured font engine will simulate the effect by slanting the upright characters to make them appear to be italic. Other systems may simulate italic with oblique and vice versa.

The differences between the italic and oblique faces is in the treatment of certain characters such as a small letter a. In one model the same letter form as is used for the upright font is drawn with its verticals tilted at an angle. In the other the letter form is still drawn tilted but an alternative shape is used. In a properly supported system with all the special font renderings available, the characters will be specially drawn so they look clean when slanted. Simply skewing the upright characters does not preserve the character shape as accurately.

The following keywords can be applied to this property:

- `normal`
- `italic`
- `oblique`

Warnings:

- Netscape Navigator does not understand the `oblique` keyword when defining style values for `fontStyle` properties.
- The MSIE browser understands both `italic` and `oblique` keywords but assumes that they both mean `italic`.

See also:	`JSSTag.fontStyle,String.italics(),String.sub(), String.sup(),style.font`

style.fontVariant (Property)

The small-caps variant of a font for a style.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.fontVariant
CSS syntax:	font-variant: *aVariant*	
Argument list:	*aVariant*	A font variant

This is a popular design trick which renders lower-case letters in upper-case but in a smaller font size. This is often called small-caps.

The following keywords can be applied to this property:

- `normal`
- `small-caps`

See also:	`style.font`

style.fontWeight (Property)

The boldness of text drawn in the current font.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.fontWeight
CSS syntax:	font-weight: *aWeight*	
Argument list:	*aWeight*	A font weight

The font weight is a continuous scale from 100 to 900. Printers can resolve the font weights to a finer granular accuracy than a screen display. Screen displays can generally cope with discriminating a difference between 100 and 200 but cannot manage any better than that due to the screen resolution.

The value can also be specified with the following keywords:

- `bold`
- `bolder`
- `lighter`
- `normal`

See also:	`JSSTag.fontWeight`, `String.bold()`, `style.font`

style.getAttribute() (Method)

A method to extract attributes from a style.

Availability:	JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.getAttribute(`*anAttribName*`)`
	-	*myStyle*`.getAttribute(`*anAttribName*`, `*aCaseSense*`)`
Argument list:	*aCaseSense*	A flag to indicate a case-sensitive lookup
	anAttribName	An attribute of a style object

This is an accessor method which is used to access named attributes of an `Element` object. Attributes are not properties in the strict sense of the word but may be accessible as if they were in some implementations.

This accessor is intended to provide a means of managing custom attributes.

You need to know the names of the attributes you want to access. If you do, then you can pass the attribute name as an argument to this method call.

The value of that attribute is returned by the method.

It would be logical to assume that attributes are named uniquely but if several share the same name, differing only in case-sensitivity, then if a case-insensitive search is used you may not retrieve the one you expect. It is likely that you'll be given the last one that occurs but this may be implementation dependent.

The case sensitive flag should be set to the Boolean `true` value to force a case-sensitive search and `false` to ignore the case of letters in the attribute name.

The following values can be passed in the optional case-sensitive flag argument:

- `0` – A case-insensitive search of property values is carried out by default. If several instances are located, then only the last is returned.
- `1` – A case-sensitive search is carried out.
- `2` – The value is returned exactly as was originally defined in the document source regardless of subsequent `setAttribute()` calls.

The result will be the value of the attribute. If the element does not have an attribute of the specified name, a `null` value is returned.

Warnings:

- If a case-sensitive search is carried out using a property name stored in a variable, you should make sure that the same setting was defined for a corresponding `setAttribute()` call. If you don't, then it is possible that the name may have a case change if the `0` value was used in the `setAttribute()` call. After that case change, the value in the variable will no longer match the property defined for the receiving object.

See also:	`Element.getAttribute(), style.setAttribute()`

style.getExpression() (Method)

An MSIE extension for managing style controls.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.getExpression(`*`aProperty`*`)`
Argument list:	*`aProperty`*	The name of a property whose expression is to be retrieved

This is part of the MSIE support for managing behaviors. With this method, you can access the value of an expression that was previously defined with the `setExpression()` method or which is defined within a style rule.

The rules that are used to construct a style are comprised of multiple expressions. You can use this method to extract the value of an expression from a style item.

Given the rule might contain a line such as:

```
width:200px
```

The `getExpression()` method can be applied to the style object containing the rule with that expression like this:

```
myStyle.getExpression("width:")
```

The value `200px` would be returned.

See also:	`Element.getExpression()`, `Element.removeExpression()`, `Element.setExpression()`, `style.removeExpression()`,`style.setExpression()`

style.height (Property)

The height of a sizing style.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.height
CSS syntax:	`height: aHeight`	
Argument list:	*aHeight*	An object height

The object space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

When used to read the height on an object, this returns a value in pixel units with the `px` suffix.

You can also use this property to change the height of a styled element.

The normal range of values specified in measurement units can be used. You can also assign the `auto` keyword to let the browser deduce the height of an object from the document source.

Warnings:

- Note that not all styled elements can be resized by assigning a new value to this property.

See also:	`JSSTag.height`, Measurement units, `style.pixelHeight`, `style.posHeight`,`style.width`

style.imeMode (Property)

An Input Method Editor mode specifier.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0

Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`style`*`.imeMode`
CSS syntax:	`ime-mode:` *`aControl`*	
Argument list:	*`aControl`*	One of the available control keywords

Input Method Editors are provided to support Asian languages such as Chinese, Korean and Japanese.

This property maintains an IME specifier value.

The following keywords are valid:

- `auto`
- `active`
- `inactive`
- `disabled`

style.important (Property)

A means of adding emphasis to an object.

Availability:	CSS level – 1 JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.important`
CSS syntax:	`!important`	

Although the `!important` style attribute is supported in CSS style sheets used by MSIE and Netscape 6.0, there is apparently no access to it from JScript at this time. If there were, then syntax rules for JavaScript would preclude the use of the exclamation mark as a part of the identifier name for the property so it would most likely be called simply `style.important`.

Although there are no values assigned to this style attribute in the CSS style sheet, it is likely that it would be a Boolean value if it were accessible from JScript.

style.item() (Method)

When the style is treated as if it were a collection, objects belonging to it can be referenced by their item numbers.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`style object`	
JavaScript syntax:	IE	`myStyle.item(anIndex)`
	IE	*`myStyle`*`.item(`*`aSelector`*`)`
	IE	*`myStyle`*`.item(`*`aSelector, anIndex`*`)`
Argument list:	*`anIndex`*	A zero based value in accordance with the `length` attribute of the style object.
	`aSelector`	A textual value that selects all matching objects

You will have to use some JavaScript to inspect the values presented by this mechanism. The following lines both yield a `FONT-SIZE` object:

```
document.all.tags("HTML")[0].currentStyle.item(0)

document.all.tags("TITLE")[0].currentStyle.item(0)
```

The style object in both cases only has a single item in its collection and so the length values for these expressions always yield `1` for a simple document:

```
document.all.tags("HTML")[0].currentStyle.length

document.all.tags("TITLE")[0].currentStyle.length
```

If you begin to explore the MSIE browser in this way, there are many undocumented properties and object references which are reasonably easy to figure out because of their names. However some are quite obscure. We have tried to cover as many as we can. Discovering undocumented method calls is far more difficult and you may need to reverse engineer the JScript interpreter to discover the names of the methods that are implemented.

See also:	`Collection.Item()`

style.layoutGrid (Property)

An MSIE extension that provides a means of laying out objects on a grid.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0
Property/method value type:	`String primitive`

JavaScript syntax:	IE	*myStyle*.layoutGrid
CSS syntax:	layout-grid: *aLayout*	
Argument list:	*aLayout*	One of the available layout grid control keywords

The layout grid is especially useful when presenting Asian text. This uses ideographic characters where each single character represents a word on its own. They need to be arranged in an orderly manner but it is quite inconvenient to build a table with one cell for each character. A layout grid accomplishes the lining up and justification of each column and row in a much more convenient manner.

This is a convenience method for specifying all of the grid control values in a single assignment.

The values for the following related properties should be used to specify this value:

- layoutGridChar
- layoutGridLine
- layoutGridMode
- layoutGridType

Only the style.layoutGridCharSpacing is not supported by this short cut.

Since the namespaces for the values that can be used in these properties clash with one another, they should be presented in the following order:

<mode> <type> <line> <char>

Refer to the topics for these individual properties for details of the available values you can use.

See also:	style.layoutGridChar, style.layoutGridCharSpacing, style.layoutGridLine, style.layoutGridMode, style.layoutGridType

style.layoutGridChar (Property)

Part of the MSIE grid layout control.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	String primitive	
JavaScript syntax:	IE	*myStyle*.layoutGridChar
CSS syntax:	layout-grid-char: *aSize*	
Argument list:	*aSize*	A size specifier for the layout grid

The grid is defined according to various metrics associated with the font or can be explicitly specified. The following keyword values can be used:

- `none`
- `auto`

When set to `auto`, the largest character in the current font is used to define the height and width of each grid cell.

The value can also be specified in measurement units or as a percentage of the parent containing object.

See also:	Measurement units, `style.layoutGrid`

style.layoutGridCharSpacing (Property)

Spacing control for the MSIE grid layout extensions.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.layoutGridCharSpacing`
CSS syntax:	`layout-grid-char-spacing:` *aValue*	
Argument list:	*aValue*	Character spacing value within the grid

This is an additional parameter to control spacing between characters in the grid.

See also:	`style.layoutGrid`

style.layoutGridLine (Property)

Additional control for the MSIE grid layout extensions.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.layoutGridLine`
CSS syntax:	`layout-grid-line:` *aControl*	
Argument list:	*aControl*	A control value

This provides a fine level of control for adjusting the spacing between lines in the grid. The values available for use can be one of the following keywords:

- `none`
- `auto`

You can also specify a value in length units or as a percentage.

See also:	Measurement units, `style.layoutGrid`

style.layoutGridMode (Property)

Mode settings for the MSIE grid layout extensions.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.layoutGridMode`
CSS syntax:	`layout-grid-mode:` *aControl*	
Argument list:	*aControl*	A specifier for the mode of operation of the layout grid

This property defines whether the layout grid uses one or two axes or neither for positioning characters. The following keywords can be used:

- `both`
- `none`
- `line`
- `char`

The default is for both the line and character spacing to be used. Using the `line` or `char` keywords signifies that only one axis is to be used for the grid.

See also:	`style.layoutGrid`

style.layoutGridType (Property)

A type selector for the MSIE grid layout extensions.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.layoutGridType
CSS syntax:	layout-grid-type: *aType*	
Argument list:	*aType*	Select a type of layout grid

There are various kinds of grid supported. An appropriate grid type is selected with one of the following keywords:

- `loose`
- `strict`
- `fixed`

The kind of grid you would select will depend on the language and kind of ideographic font being used.

See also:	`style.layoutGrid`

style.left (Property)

A positioning reference point.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.left
CSS syntax:	left: *aPosition*	
Argument list:	*aPosition*	A left coordinate value

A CSS-P positioning style attribute that controls the location of an element relative to its containing parent element. The left edges of the two elements are used as the reference points.

The value can be specified in the usual pixel or fractional em-dash measurement units or the `auto` keyword can be used to let the browser do the positioning itself.

The exact positioning is affected by settings for padding, border, margin, and whether the position property is set to absolute or relative.

See also:	Measurement units, `style.bottom`, `style.pixelLeft`, `style.posLeft`, `style.right`, `style.top`

style.length (Property)

The `style` object can be treated as if it were a collection. This property indicates the number of objects that are in the collection.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.length`

Property attributes:

`ReadOnly`.

Refer to:

`Collection.length`

style.letterSpacing (Property)

The letter spacing of text in a style.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyle`*`.letterSpacing`
CSS syntax:	`letter-spacing:` *`aValue`*	
Argument list:	*`aValue`*	A letter spacing factor

The text can be spaced out to fill a larger space or provide some emphasis. This property may also condense the spacing by specifying a negative value. The value specified is added to the normal spacing between characters.

You can use the pixel or em-dash length values and a floating point value can be specified for the em-dash measurement units. The `normal` keyword is reserved for restoring the value back to its default setting.

The `auto` keyword is also defined in the CSS specification and can be assigned to the property to let the browser assume control.

See also:	Measurement units

style.lineBreak (Property)

Line breaking control style for Japanese text layouts.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.lineBreak`
CSS syntax:	`line-break: aControl`	
Argument list:	*`aControl`*	One of the available keywords

This property controls the way that line breaks are handled for Japanese text.

The following keywords are supported:

- `normal`
- `strict`

style.lineHeight (Property)

Defines the height of a box that contains a line of text. This is the distance between the base lines of two adjacent lines of text.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.lineHeight
CSS syntax:	line-height: *aLineHeight*	
Argument list:	*aLineHeight*	A line height value

This property provides a way to adjust the leading or spacing between the lines of text. The value in this property describes the height of a bounding box that surrounds a single line of text.

This property can have a length value specified in pixels or floating point multiples of an em-dash in the current font. The `normal` keyword can also be used to restore the default behavior.

See also:	`JSSTag.lineHeight,style.font`

style.listStyle (Property)

A shortcut property for defining several list style attributes in a single assignment.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.listStyle
CSS syntax:	list-style: *aType aPosition anImage*	
Argument list:	*aType*	A list type
	aPosition	A bullet position control
	anImage	A bullet image URL

This property supports the assignment of values to the following properties in one single operation:

- `style.listStyleImage`
- `style.listStylePosition`
- `style.listStyleType`

Refer to the topics discussing each individual property for details of the values you can use.

The values can be specified in any order or combination because their namespaces do not collide.

See also:	`style.counterIncrement,style.listStyleImage, style.listStylePosition,style.listStyleType`

style.listStyleImage (Property)

A URL for an image resource to be used for bullets in a list style.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.listStyleImage`
CSS syntax:	`list-style-image:` *anImage*	
Argument list:	*anImage*	A bullet image URL

You can modify the image used as a leading bullet in a list by defining a URL here. The image will be fetched by the browser and then placed into the page at the front of each list item.

Note that this property is cascaded down and inherited by other elements so if you want to create sub-lists you will need to redefine it in child list styles.

The `none` keyword will restore the list bullet to its default appearance.

See also:	`style.listStyle,style.listStyleType`

style.listStylePosition (Property)

A position control for a list marker.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.listStylePosition`
CSS syntax:	`list-style-position:` *aPosition*	
Argument list:	*aPosition*	A bullet position control

The list item marker can be positioned inside or outside the extent rectangle for the list item. The following keywords can be used with this property:

- `inside`
- `outside`

The bullet placement and indentation of the list item are left-justified differently according to the value of this property.

Note that there is some relationship between the behavior of this property and the alignment settings for the element in the list item.

See also:	Measurement units, `style.listStyle`

style.listStyleType (Property)

The type of list presentation marker for an ordered (`<OL>`) or unordered (`<UL>`) list.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyle`*`.listStyleType`
CSS syntax:	`list-style-stype:` *`aType`*	
Argument list:	*`aType`*	A list type

You can define the way that list items are enumerated on the page. The specific details depend on whether the list item is a member of an ordered list (`<OL>`) or an unordered list (`<UL>`).

The following keywords are appropriate for use with unordered lists:

- `circle`
- `disc`
- `square`

The default setting for an unordered list will be a disc.

The following keywords are appropriate for an ordered list:

- `decimal`
- `lower-alpha`
- `lower-roman`
- `upper-alpha`
- `upper-roman`

The default setting for an ordered list is decimal.

The keywords for this property correspond with the values defined in `UL.type` and `OL.type` object properties. Although the notation is different the displayed artefacts will be the same.

This display control property is completely overridden if the `listStyleImage` property is set to a URL for an image. As long as the `listStyleImage` property is set to "`none`" or an empty ("") string this property comes into play.

See also:	`JSSTag.listStyleType`, `OL.type`, `style.counterIncrement`, `style.listStyle`, `style.listStyleImage`, `UL.type`

style.margin (Property)

The margin around a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.margin
CSS syntax:	margin: *aWidth* ...	
Argument list:	*aWidth*	Up to four width values can be specified

This property lets you change the width of one or more of the margins around an object.

If you specify only one value, all four margins are set. Two values define the top and bottom with the first, and the second then applies to the left and right. If three values are specified, the first controls the top, the second controls both left and right,while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order: top, right, bottom and left.

Width values can be specified using pixel values, em-dash units or percentages of the next outermost parent object. The `auto` keyword lets the browser decide for itself.

Warnings:

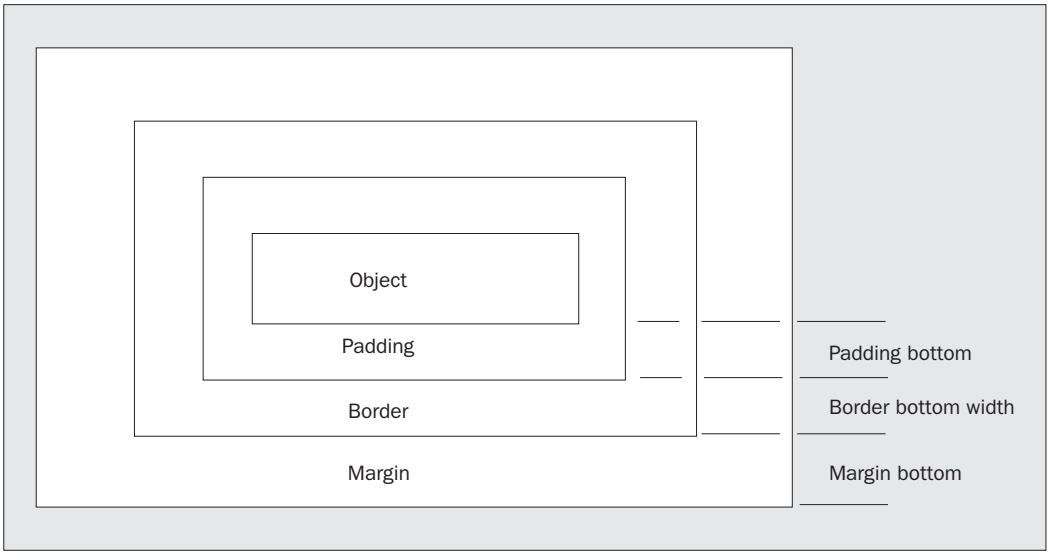
- Although both Netscape Navigator and MSIE support padding and margin control properties. It is not clear from the documentation sources whether they are simply different names for the same thing. Padding is a space that is placed outside the content rectangle of the element's extent. Margins also are not calculated as part of the element's width and height so they appear to be the same thing, although functionally they should be contained inside the extent rectangle if they behave in a manner consistent with the way that margins are generally assumed to work.
- Be wary of using margins and padding in case they undergo a functional change in the future.

See also:	`JSSTag.margins()`, Measurement units, `style.border`, `style.borderWidth`, `style.padding`

style.marginBottom (Property)

The thickness of the bottom margin of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.marginBottom
CSS syntax:	margin-bottom: *aWidth*	
Argument list:	*aWidth*	A margin width value



See also:	BODY.bottomMargin, JSSTag.marginBottom, style.margin

style.marginLeft (Property)

The thickness of the left margin of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.marginLeft
CSS syntax:	margin-left: *aWidth*	
Argument list:	*aWidth*	A margin width value
See also:	`BODY.leftMargin,JSSTag.marginLeft,style.margin`	

style.marginRight (Property)

The thickness of the right margin of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.marginRight
CSS syntax:	margin-right: *aWidth*	
Argument list:	*aWidth*	A margin width value
See also:	`BODY.rightMargin,JSSTag.marginRight,style.margin`	

style.marginTop (Property)

The thickness of the top margin of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.marginTop
CSS syntax:	margin-top: *aWidth*	
Argument list:	*aWidth*	A margin width value

See also:	`BODY.topMargin`, `JSSTag.marginTop`, `style.margin`

style.markerOffset (Property)

A spacing distance between the list item marker and the list item content.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.markerOffset
CSS syntax:	marker-offset: *aValue*	
Argument list:	*aValue*	A control for marker offsets

The distance between the marker and the text it is marking can be specified in length units. That is pixels or proportions of an em-dash. The `auto` keyword can be used to indicate default behavior, or the `inherit` keyword can indicate that the value should be defined in a parent element object.

See also:	Measurement units

style.marks (Property)

A control attribute that determines whether crop marks should be added to the page when it is printed.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.marks
CSS syntax:	marks: *aType*	
Argument list:	*aType*	A marker type

This is usually functionality that is reserved for printed pages. It is applied within the `@page` rule which is a special rule in the style sheet for laying out pages.

The following mark types are supported:

- `crop`
- `cross`

Crop marks are used for trimming while cross marks are used for aligning each print color in the press.

style.maxHeight (Property)

Defines the maximum height of a styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.maxHeight
CSS syntax:	max-height: *aValue*	
Argument list:	*aValue*	A maximum height value

This property provides a means of specifying the maximum height of an element so that it cannot get any bigger, regardless of the implications of the document flow and automatic browser formatting.

The value is expressed in the usual measurement units (either pixels or fractions of an em-dash). You can also use the percentage of the containing element as a maximum size.

The size of an element can be constrained using the minimum/maximum extents for the height and width. This ensures that the element's size will fall within the permitted bounds but yet still have some flexibility to allow the document flow to respond to window sizing.

See also:	Measurement units, `style.maxWidth`, `style.minHeight`

style.maxWidth (Property)

Defines the maximum width of a styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyle`*`.maxWidth`
CSS syntax:	`max-width: `*`aValue`*	
Argument list:	*`aValue`*	A maximum width value

This property provides a means of specifying the maximum width of an element so that it cannot get any bigger, regardless of the implications of the document flow and automatic browser formatting.

The value is expressed in the usual measurement units (either pixels or fractions of an em-dash). You can also use the percentage of the containing element as a maximum size.

The size of an element can be constrained using the minimum/maximum extents for the height and width. This ensures that the element's size will fall within the permitted bounds, but yet still have some flexibility to allow the document flow to respond to window sizing.

See also:	Measurement units, `style.maxHeight`, `style.minWidth`

style.minHeight (Property)

Defines the minimum height of a styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.minHeight
CSS syntax:	min-height: *aValue*	
Argument list:	*aValue*	A minimum height value

This property provides a means of specifying the minimum height of an element so that it cannot get any smaller, regardless of the implications of the document flow and automatic browser formatting.

The value is expressed in the usual measurement units (either pixels or fractions of an em-dash). You can also use the percentage of the containing element as a maximum size.

The size of an element can be constrained using the minimum/maximum extents for the height and width. This ensures that the element's size will fall within the permitted bounds, but yet still have some flexibility to allow the document flow to respond to window sizing.

See also:	Measurement units, `style.maxHeight`, `style.minWidth`

style.minWidth (Property)

Defines the minimum width of a styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.minWidth
CSS syntax:	min-width: *aValue*	
Argument list:	*aValue*	A minimum width value

This property provides a means of specifying the minimum width of an element so that it cannot get any smaller, regardless of the implications of the document flow and automatic browser formatting.

The value is expressed in the usual measurement units (either pixels or fractions of an em-dash). You can also use the percentage of the containing element as a maximum size.

The size of an element can be constrained using the minimum/maximum extents for the height and width. This ensures that the element's size will fall within the permitted bounds but yet still have some flexibility to allow the document flow to respond to window sizing.

See also:	Measurement units, `style.maxWidth`, `style.minHeight`

style.orphans (Property)

Defines the minimum number of lines of a paragraph of text that must be visible at the bottom of a page when a page break is present. This is most likely to occur when printing documents.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.orphans
CSS syntax:	orphans: *aCount*	
Argument list:	*aCount*	A number of lines for an orphan count

Widows and orphans are fragments of text that appear to be formatted incorrectly when a paragraph of text spans a page break.

Windows and orphans are usually controlled together and the usual technique is to specify that an entire paragraph should be kept on the same page. This forces the paragraph to be taken over to the next page in its entirety, even if the flow requires just a single word to be taken over.

The CSS styling controls allow a finer level of control in that you can allow for a paragraph to be split across a page boundary, but specify a lower limit on the number of lines that must be kept on a single page.

This is fine in principle, but there can be some contention for the right layout when a very short paragraph is spanning a page break. This will generally be solved simply by forcing the page break to happen before the paragraph causing the whole paragraph to be carried over to the next page.

An orphan is that fragment of text that is left at the bottom of a page when a paragraph encloses a page break. It is the topmost few lines of the paragraph. The integer value in this property controls the minimum number of lines that must be present, otherwise the paragraph will be taken over entirely to the next page.

See also:	`style.pageBreakAfter`, `style.size`, `style.widows`

style.outline (Property)

A shortcut attribute for defining all the outline settings together.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.outline
CSS syntax:	outline: *aColour aStyle aWidth* outline: aControl	
Argument list:	*aColour*	An outline color value
	aStyle	A style of outlining
	aWidth	A thickness of the outline
	aControl	A control over inheritance

An outline is like a border but it is drawn within the extent of the object. It is intended to be drawn and then removed and is useful for those times when you want to border something without the border taking up any space. Its width is measured inwards from the outer edge of an object.

This property is a means of defining the following style properties with a single assignment:

- `style.outlineColor`
- `style.outlineStyle`
- `style.outlineWidth`

Because the namespaces do not collide, they can be specified in any order.

See also:	`style.border,style.outlineColor, style.outlineStyle,style.outlineWidth`

style.outlineColor (Property)

The color of a border outline around the styled element.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myStyle*.outlineColor
CSS syntax:	outline-color: *aColor*	
Argument list:	*aColor*	An outline color value

This property lets you change the color of one or more of the outlines around an object.

If you specify only one value, all four outlines are set. Two values define the top and bottom with the first, and the second then applies to the left and right. If three values are specified, the first controls the top, the second controls both left and right, while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order: top, right, bottom and left.

Color values can be specified using symbolic names, `rgb()` functions or hash delimited hex values.

See also:	Color names, Color value, `rgb()`, `style.borderColor`, `style.outline`

style.outlineStyle (Property)

The border style for an outline around a styled element.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.outlineStyle
CSS syntax:	outline-style: *aStyle*	
Argument list:	*aStyle*	A style of outlining

This property lets you change the style of one or more of the outlines around an object.

If you specify only one value, all four outlines are set. Two values define the top and bottom with the first, and the second then applies to the left and right. If three values are specified, the first controls the top, the second controls both left and right, while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order: top, right, bottom and left.

Style values can be specified using symbolic names as follows:

- `solid`
- `dashed`
- `dotted`

- double
- inset
- outset
- groove
- ridge
- hidden
- none

See also:	style.borderStyle, style.outline

style.outlineWidth (Property)

The width of a border that outlines a styled element.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.outlineWidth
CSS syntax:	outline-width: *aWidth*	
Argument list:	*aWidth*	A thickness of the outline

This property lets you change the width of one or more of the outlines around an object.

If you specify only one value, all four outlines are set. Two values define the top and bottom with the first, and the second then applies to the left and right. If three values are specified, the first controls the top, the second controls both left and right, while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order: top, right, bottom and left.

Width values can be specified using symbolic names or pixel values. The following symbolic names are available:

- thin
- medium
- thick

See also:	style.borderWidth, style.outline

style.overflow (Property)

The overflow style that defines how to display content that is too large to fix the element's stated box size.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.overflow
CSS syntax:	overflow: *aType*	
Argument list:	*aType*	An overflow type specifier

Sometimes you may add content to an element and cause the element to overflow its bounding extent rectangle. This property helps the browser determine what it should do with that overflowed content.

The following keywords may be applied to this property:

- `auto`
- `hidden`
- `scroll`
- `visible`

The general approach to handling overflow is to wrap the content within the width and increase the height to accommodate the flow. This may not be appropriate for some objects whose width formatting is fixed. This would be the case with a `PRE` object for example.

The `visible` setting allows the width to be expanded to accommodate the object in attempt to keep it all visible at once.

The `hidden` setting preserves the settings for the height and width. It may rearrange the content to better fill the extent rectangle, but it may clip the content within the extent rectangle so that the object retains the size it had already been defined to be presented with.

The `scroll` setting allows for scroll bars to be placed within the extent rectangle of the object itself if the content exceeds the present settings of the height and width extent rectangle. This behavior is not consistent across all platforms.

The `auto` setting displays scrollbars within the extent rectangle of the object if necessary, but appears to be functionally identical to the scroll value being defined. Again, this is not consistently supported across the platforms.

The workings of this property may be affected by the value defined for the `position` property. That is, this overflow technique is applicable to absolutely positioned elements.

Warnings:

- Some versions of the MSIE browser on the Macintosh platform cannot cope with a `scroll` or `auto` attribute in this property. The scrollbars simply don't appear.

See also:	`PRE` object, `style.position`

style.overflowX (Property)

A definition of how to handle horizontally overflowing content.

Availability:	JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.overflowX
CSS syntax:	`overflow-x:` *aType*	
Argument list:	*aType*	An overflow type specifier

Refer to:

`style.overflow`

style.overflowY (Property)

A definition of how to handle vertically overflowing content.

Availability:	JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.overflowY
CSS syntax:	`overflow-y:` *aType*	
Argument list:	*aType*	An overflow type specifier

Refer to:

`style.overflow`

style.padding (Property)

A shortcut means of specifying all the padding attributes for a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.padding
CSS syntax:	padding: *aWidth* ...	
Argument list:	*aWidth*	Up to four padding widths

This property lets you change the width of one or more of the padding regions around an object.

If you specify only one value, all four padding regions are set. Two values define the top and bottom with the first, and the second then applies to the left and right,. If three values are specified, the first controls the top, the second controls both left and right while the third controls the bottom edge. When all four values are specified, they are assumed to be in the order, top, right, bottom and left.

Width values can be specified using pixel values, em-dash units or percentages of the next outermost parent object. The `auto` keyword lets the browser decide for itself.

You can also specify the values independently with the following properties:

- `style.paddingBottom`
- `style.paddingLeft`
- `style.paddingRight`
- `style.paddingTop`

Warnings:

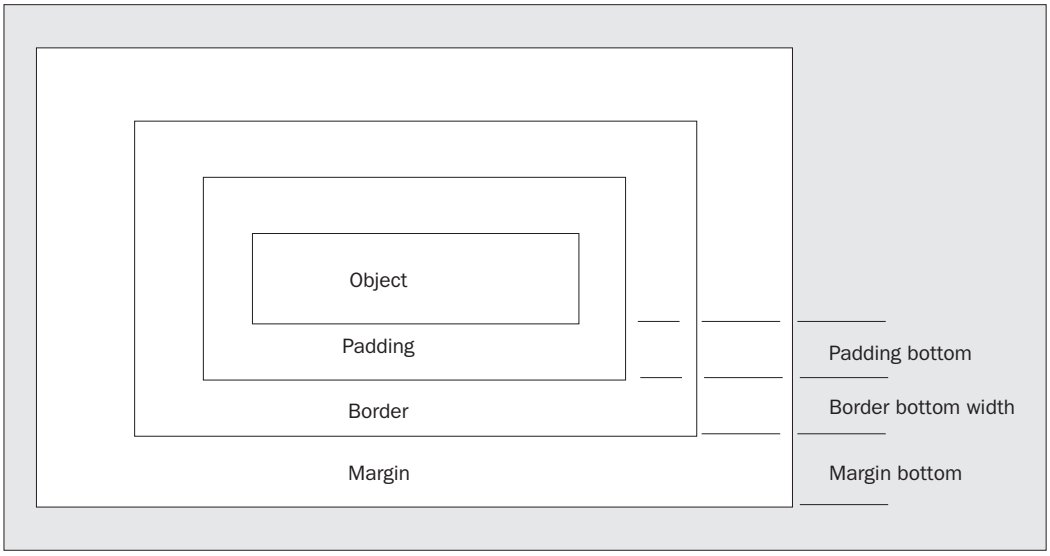
- Although both Netscape Navigator and MSIE support padding and margin control properties, it is not clear from the documentation sources whether they are simply different names for the same thing. Padding is a space that is placed outside the content rectangle of the element's extent. Margins also are not calculated as part of the element's width and height so they appear to be the same thing, although functionally they should be contained inside the extent rectangle, if they behave in a manner consistent with the way that margins are generally assumed to work.
- Be wary of using margins and padding in case the way they are implemented undergoes a functional change in the future.

See also:	`JSSTag.paddings()`, Measurement units, style.border, `style.margin`

style.paddingBottom (Property)

A value for the thickness of the padding space at the bottom of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.paddingBottom
CSS syntax:	padding-bottom: *aWidth*	
Argument list:	*aWidth*	A padding width value



See also:	`JSSTag.paddingBottom`, Measurement units, `style.padding`, `style.margin`

style.paddingLeft (Property)

A value for the thickness of the padding space to the left of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.paddingLeft
CSS syntax:	padding-left: *aWidth*	
Argument list:	*aWidth*	A padding width value
See also:	`JSSTag.paddingLeft`, Measurement units, `style.padding`, `style.margin`	

style.paddingRight (Property)

A value for the thickness of the padding space to the right of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.paddingRight
CSS syntax:	padding-right: *aWidth*	
Argument list:	*aWidth*	A padding width value
See also:	`JSSTag.paddingRight`, Measurement units, `style.padding`, `style.margin`	

style.paddingTop (Property)

A value for the thickness of the padding space at the top of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.paddingTop
CSS syntax:	padding-top: *aWidth*	
Argument list:	*aWidth*	A padding width value

See also:	`JSSTag.paddingTop`, Measurement units, `style.padding`, `style.margin`

Refer to:

style.page (Property)

A means of placing a styled element onto a particular page. If necessary a page break will be created to accommodate the required location.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.page
CSS syntax:	page: *aValue*	
Argument list:	*aValue*	A page identifier

This is part of the printing support in CSS. The property can accept either the `auto` keyword, which allows the element to be printed on the current page. It can also have a named page layout cited as an identifier value. That named layout must have been created earlier in the style sheet.

When this style is deployed, one of two possibilities pertains. It names a page layout which is the current layout being rendered or some other layout that is currently being written on. If the identifier corresponds to the layout currently being rendered, then the element is simply added to the page unless the page is full and then a new page is created in the normal way. If the page currently being rendered is of another type, that is its layout is different, then a new page is started and the element is the first item placed on it.

See also:	`style.size`

style.pageBreakAfter (Property)

The placement of a page break after the styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyle`*`.pageBreakAfter`
CSS syntax:	`page-break-after:` *`aSwitch`*	
Argument list:	*`aSwitch`*	A page break control value

This is part of the support for printing pages. A browser doesn't need to mark the breaks between pages because its pages are of unlimited length and width when the scrollbars are present. You need this for printouts, though, because the paper has a finite size.

The following keywords can be applied to this property:

- `auto`
- `left`
- `right`
- `always`
- `avoid`
- `inherit`

These attributes are part of the CSS level 2 specification, which is not yet fully supported in most browsers.

The `auto` setting allows the browser to use its best opinion on whether to place a page break after the element.

The `left` and `right` keywords provide a swinging format capability based on whether the page is an odd or even page (by implication odd number pages are rightwards). This allows page breaks to be placed depending on whether element flows onto a right or leftwards facing page.

The `always` value requires that a page break happens after this element on every occasion regardless of whether the page is odd or even.

The CSS level 2 standard allows for the property to contain the `avoid` value. This is not yet supported in all browsers and simply inhibits page breaks after the element.

The above all applies equally to the `pageBreakBefore` property apart from noting that the page break is placed prior to the object rather than after it.

The `pageBreakInside` property only honors the `auto`, `avoid` and `inherit` keywords.

Warnings:

- This property does not work consistently across browsers and platforms. Unusually, it works better in MSIE for Macintosh than Windows.

See also:	`style.orphans`, `style.size`, `style.widows`

style.pageBreakBefore (Property)

The placement of a page break before the styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.pageBreakBefore`
CSS syntax:	`page-break-before:` *aSwitch*	
Argument list:	*aSwitch*	A page break control value

See also:	`style.size`, `style.pageBreakAfter`

style.pageBreakInside (Property)

Indicates whether a page break can occur inside the element if necessary.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.pageBreakInside
CSS syntax:	page-break-inside: *aSwitch*	
Argument list:	*aSwitch*	A page break control value

See also:	style.page, style.size, style.pageBreakAfter

style.pause (Property)

Part of the aural style control suite that defines a momentary pause before or after an item is spoken.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.pause
CSS syntax:	pause: *aTime*	
Argument list:	*aTime*	A pause duration between speeches

This is a shortcut property for defining the `pauseBefore` and `pauseAfter` values in a single assignment. The two values should be space separated. If you only specify one, it will be assigned to both properties. If you specify two values, the first is used as `pauseBefore` and the second as `pauseAfter`.

The values are specified as time units. These are a floating point value with either the `ms` or `s` suffix to denote the unit of measure.

You can specify a percentage value which is used to calculate a proportion of the time taken to speak a single word at the words-per-minute rate defined in the `speechRate` property.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, Measurement units, `style.pauseAfter`, `style.pauseBefore`, `style.speechRate`

style.pauseAfter (Property)

Part of the aural style control suite that defines a momentary pause after an item is spoken.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	`myStyle.pauseAfter`
CSS syntax:	`pause-after: aTime`	
Argument list:	*`aTime`*	A pause duration after speech

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, `style.pause`

style.pauseBefore (Property)

Part of the aural style control suite that defines a momentary pause before an item is spoken.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*`myStyle`*`.pauseBefore`
CSS syntax:	`pause-before: aTime`	
Argument list:	*`aTime`*	A pause duration before speech

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, `style.pause`

style.pitch (Property)

Part of the aural style control suite that defines the average pitch frequency of the voice used to speak the text.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.pitch
CSS syntax:	pitch: *aFrequency*	
Argument list:	*aFrequency*	A frequency to pitch the voice

You can specify the value in hertz or kilohertz. The measurement units specifier for these would be `Hz` or `kHz`. The numeric values would normally be an integer for the Hz units and a floating point value for the kHz units. If you prefer, the following keywords can be used:

- `x-low`
- `low`
- `medium`
- `high`
- `x-high`

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, Measurement units

style.pitchRange (Property)

Part of the aural style control suite that defines the variance of the spoken voice pitch when it is capable of rendering emphasis by changing its average pitch.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.pitchRange
CSS syntax:	pitch-range: *aRange*	
Argument list:	*aRange*	A range of pitch variation

The pitch range allows for the voice to raise and fall during speech to make it sound more interesting.

The value of `0` can be used to prevent any pitch change during speech. This would cause speech to use a monotone and sound very dull. A value of `50` would sound normal while a value of `100` would sound somewhat excited.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, `style.richness`

style.pixelBottom (Property)

The location of the styled element measured in pixel units within the page.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.pixelBottom`
CSS syntax:	`pixel-bottom:` *aValue*	
Argument list:	*aValue*	A positioning value

This object was added in version 5 of MSIE to complete the set of positioning properties already available with the `style` object in earlier versions of MSIE.

Objects can be positioned on screen with this property. This value defines the location of the bottom edge of a styled element. The bottom edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

This property will be useful as an alternative to the `style.bottom` property which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in pixel measurement units only. This is regardless of the settings of the measurement units in the CSS attribute. Refer to the `style.posBottom` property for a more generalized value that can be operated on with different measurement units.

See also:	Measurement units, `style.bottom`, `style.posBottom`

style.pixelHeight (Property)

The pixel height of the styled element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.pixelHeight`
CSS syntax:	`pixel-height:` *aHeight*	
Argument list:	*aHeight*	A sizing value

Although this is a read and write property, it is most likely to be useful for measuring the height of an object to get a numeric value without the trailing `px` suffix that you get when enquiring the `style.height` value.

The value in this property will be an integer defined in pixel measurement units only. This is regardless of the settings of the measurement units in the CSS attribute. Refer to the `style.posHeight` property for a more generalized value that can be operated on with different measurement units.

See also:	Measurement units, `style.height`, `style.pixelWidth`, `style.posHeight`

style.pixelLeft (Property)

The pixel position of the left edge of the styled element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.pixelLeft`
CSS syntax:	`pixel-left:` *aValue*	
Argument list:	*aValue*	A positioning value

Objects can be positioned on screen with this property. This value defines the location of the left edge of a styled element. The left edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

The value in this property should be reflected in the `Element.offsetLeft` property.

This property will be useful as an alternative to the `style.left` property which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in pixel measurement units only. This is regardless of the settings of the measurement units in the CSS attribute. Refer to the `style.posLeft` property for a more generalized value that can be operated on with different measurement units.

See also:	`Element.offsetLeft`, `Layer.left`, Measurement units, `style.left`, `style.posLeft`

style.pixelRight (Property)

The pixel position of the right edge of the styled element.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.pixelRight`
CSS syntax:	`pixel-right:` *aValue*	
Argument list:	*aValue*	A positioning value

This object was added in version 5 of MSIE to complete the set of positioning properties already available with the `style` object in earlier versions of MSIE.

Objects can be positioned on screen with this property. This value defines the location of the right edge of a styled element. The right edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

This property will be useful as an alternative to the `style.right` property, which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in pixel measurement units only. This is regardless of the settings of the measurement units in the CSS attribute. Refer to the `style.posRight` property for a more generalized value that can be operated on with different measurement units.

See also:	Measurement units, `style.posRight`, `style.right`

style.pixelTop (Property)

The pixel position of the top edge of the styled element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.pixelTop
CSS syntax:	pixel-top: *aValue*	
Argument list:	*aValue*	A positioning value

Objects can be positioned on screen with this property. This value defines the location of the top edge of a styled element. The top edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

The value in this property should be reflected in the `Element.offsetTop` property.

This property will be useful as an alternative to the `style.top` property which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in pixel measurement units only. This is regardless of the settings of the measurement units in the CSS attribute. Refer to the `style.posTop` property for a more generalized value that can be operated on with different measurement units.

See also:	`Element.offsetTop`, `Layer.top`, Measurement units, `style.posTop`, `style.top`

style.pixelWidth (Property)

The pixel width of the styled element.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Opera – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	myStyle.pixelWidth
CSS syntax:	pixel-width: *aWidth*	
Argument list:	*aWidth*	A sizing value

Although this is a read and write property, it is most likely to be useful for measuring the width of an object to get a numeric value without the trailing `px` suffix that you get when enquiring the `style.width` value.

The value in this property will be an integer defined in pixel measurement units only. This is regardless of the settings of the measurement units in the CSS attribute. Refer to the `style.posWidth` property for a more generalized value that can be operated on with different measurement units.

See also:	Measurement units, `style.pixelHeight`, `style.posHeight`, `style.posWidth`, `style.width`

style.playDuring (Property)

Part of the aural style control suite that controls the mix between foreground and background sound effects.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*`.playDuring`
CSS syntax:	`play-during:` *aControl*	
Argument list:	*aControl*	A mix control for background sounds

Foreground and background sound effects need to be controlled carefully and blended properly, so that background noises such as music and sound effects do not distract from the speech.

The values for this comprise a URI value and an optional controlling parameter. The value should be space-separated.

The URI specifies an audio file to be downloaded and played in the background.

The following keywords control how the background sound is mixed with any parent object's sound effects:

- `mix`
- `repeat`
- `auto`
- `none`

The `mix` keyword combines any sounds already playing as defined by parent elements.

The `repeat` keyword allows for the background sound to be looped as required to extend it to the same length as the spoken content.

The `auto` value allows the parent elements sound effects to be played without being interrupted by this elements sounds.

The `none` value turns off any sound effects inherited from parent objects only playing sounds controlled by this object.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.posBottom (Property)

A measurement unit independent positioning control property.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.posBottom`
CSS syntax:	`pos-bottom: aValue`	
Argument list:	*aValue*	A positioning value

This object was added in version 5 of MSIE to complete the set of positioning properties already available with the `style` object in earlier versions of MSIE.

Objects can be positioned on screen with this property. This value defines the location of the bottom edge of a styled element. The bottom edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

This property will be useful as an alternative to the `style.bottom` property, which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in the measurement units defined by the CSS attribute. Refer to the `style.pixelBottom` property for a more specialized value that can be operated on unambiguously in pixel measurement units.

See also:	Measurement units, `style.bottom`, `style.pixelBottom`

style.posHeight (Property)

A measurement unit independent size control property.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.posHeight`
CSS syntax:	`pos-height:` *aHeight*	
Argument list:	*aHeight*	A sizing value

Although this is a read and write property, it is most likely to be useful for measuring the height of an object to get a numeric value without the trailing measurement unit suffix that you get when enquiring the `style.width` value.

The value in this property will be an integer defined in CSS attribute defined measurement units only. Refer to the `style.pixelHeight` property for a more specialized value that can be operated on unambiguously using pixel values.

See also:	Measurement units, `style.height`, `style.pixelHeight`, `style.pixelWidth`, `style.posWidth`

style.position (Property)

A flag to indicate relative or absolute positioning of an element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.position = "absolute"`
	-	*myStyle*`.position = "relative"`
CSS syntax:	`position:` *aControl*	
Argument list:	*aControl*	A positioning control

In MSIE, this value returns a floating point value reflecting the current value of the position attribute for the `styleSheet` object.

You can define other values for this according to the available keywords:

- `absolute`
- `relative`
- `fixed`
- `normal`
- `static`

The `absolute` keyword applies to element objects that are positioned with respect to the document boundaries.

The `relative` keyword applies to element objects that are positioned with respect to a parent or containing element object.

The `fixed` keyword is defined in the CSS standard to mean that an element object should be positioned with respect to the display window.

The `normal` keyword is equivalent to the CSS defined static keyword and applies to objects whose position is controlled by the text flow. These can be block or inline structured objects.

Warnings:

- `Element` objects in Netscape Navigator 4 are positioned by means of the layer capabilities of that version of the browser. Any object that has its position property set to the absolute value will need to be managed in a layer of its own, so that it can be positioned independently of any other objects in the page. These layers are accessible via the `layers[]` property of the document.
- As of Netscape 6.0, layers are completely deprecated and have not been implemented. If you use layers at all, your pages are going to break.
- The `style.position` property is not supported consistently across browsers. Nor is it supported according to the defined default values in the CSS standard.
- The fixed keyword is not yet properly supported in MSIE.
- Netscape 6.0 does not support the `normal` keyword although it supports the functionally identical `static` keyword.

See also:	`Document.layers[]`, Dynamic positioning, Measurement units, `style.overflow`

Property attributes:

`ReadOnly`.

style.posLeft (Property)

A measurement unit independent positioning control property.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.posLeft`
CSS syntax:	`pos-left:` *`aValue`*	
Argument list:	*`aValue`*	A positioning value

Objects can be positioned on screen with this property. This value defines the location of the left edge of a styled element. The left edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

This property will be useful as an alternative to the `style.left` property which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in the measurement units defined by the CSS attribute. Refer to the `style.pixelLeft` property for a more specialized value that can be operated on unambiguously in pixel measurement units.

See also:	`Element.offsetLeft`, `Layer.left`, Measurement units, `style.left`, `style.pixelLeft`

style.posRight (Property)

A measurement unit independent positioning control property.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.posRight`
CSS syntax:	`pos-right:` *aValue*	
Argument list:	*aValue*	A positioning value

This object was added in version 5 of MSIE to complete the set of positioning properties already available with the `style` object in earlier versions of MSIE.

Objects can be positioned on screen with this property. This value defines the location of the right edge of a styled element. The right edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

This property will be useful as an alternative to the `style.right` property which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in the measurement units defined by the CSS attribute. Refer to the `style.pixelRight` property for a more specialized value that can be operated on unambiguously in pixel measurement units.

See also:	Measurement units, `style.pixelRight`, `style.right`

style.posTop (Property)

A measurement unit independent positioning control property.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.posTop`

CSS syntax:	pos-top: *aValue*	
Argument list:	*aValue*	A positioning value

Objects can be positioned on screen with this property. This value defines the location of the top edge of a styled element. The top edge includes padding, borders and margins around the object as well as its content. The position is located with reference to the next outermost (parent) container.

This property will be useful as an alternative to the `style.top` property, which includes the measurement units when a value is read back from it.

The value in this property will be an integer defined in the measurement units defined by the CSS attribute. Refer to the `style.pixelTop` property for a more specialized value that can be operated on unambiguously in pixel measurement units.

See also:	`Element.offsetTop`, `Layer.top`, Measurement units, `style.pixelTop`, `style.top`

style.posWidth (Property)

A measurement unit independent size control property.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.posWidth
CSS syntax:	pos-width: *aWidth*	
Argument list:	*aWidth*	A sizing value

Although this is a read and write property, it is most likely to be useful for measuring the width of an object to get a numeric value without the trailing measurement unit suffix that you get when enquiring the `style.width` value.

The value in this property will be an integer defined in CSS attribute defined measurement units only. Refer to the `style.pixelWidth` property for a more specialized value that can be operated on unambiguously using pixel values.

See also:	Measurement units, `style.pixelWidth`, `style.posHeight`, `style.width`

style.quotes (Property)

A list of quotation marks to use for progressively quoted content that may have nested quotation marks.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.quotes
CSS syntax:	`quotes: aValue`	
Argument list:	*aValue*	A string to use for quoting text

If you specify quotes, you should indicate pairs of quotes. Each quote should be enclosed it its own set of delimiters, which then allows multiple characters to be used for each quotation symbol.

You can alternatively use the value `none` or `inherit`.

style.removeExpression() (Method)

An MSIE extension for managing style controls.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myStyle*.removeExpression(*aProperty*)
Argument list:	*aProperty*	The name of the property whose expression is to be removed

This is part of the MSIE support for managing style expressions. With this method, you can remove an expression that was previously defined with the `setExpression()` method.

See also:	`Element.getExpression(),`
	`Element.removeExpression(),`
	`Element.setExpression(),style.getExpression(),`
	`style.setExpression()`

style.renderingIntent (Property)

An MSIE extension to control the rendering of the page. This is part of the color modeling and preservation of true color representations.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.renderingIntent`
CSS syntax:	`rendering-intent:` *aControl*	
Argument list:	*aControl*	A rendering intent specifier value

As yet, this is still being worked on. Early drafts suggest the following keywords will be available:

- `auto`
- `perceptual`
- `relative-colorimetric`
- `saturation`
- `absolute-colorimetric`

See also:	Color names, Color value, `style.color`, `style.colorProfile`

style.richness (Property)

Part of the aural style control suite that defines the forcefulness of the spoken voice.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*`.richness`
CSS syntax:	`richness:` *aValue*	
Argument list:	*aValue*	A voice quality control value

This control affects the animation, sometimes called brightness or stridency, in the voice.

The value `50` is used as the default and with this setting the voice will sound as if it's normal.

Reducing this value makes the voice more mellow and soft. Extremely low values might almost be whispered.

Increasing value to `100` makes the voice very forceful, almost shouting.

The value can be specified as a floating point value to control the richness very finely.

The behavior of the spoken voice when controlled by the aural style properties does very much depend on the in-built speech capabilities of the platform. The Macintosh already supports a very versatile and realistic voice synthesizer as an integral component of its operating system. It would be interesting to experiment with this if it were integrated with a usable browser.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, `style.pitchRange`

style.right (Property)

A positioning reference point.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.right`
CSS syntax:	`right:` *aPosition*	
Argument list:	*aPosition*	A right coordinate value

A CSS-P positioning style attribute that controls the location of an element relative to its containing parent element. The right edges of the two elements are used as the reference points.

The value can specified in the usual pixel or fractional em-dash measurement units or the `auto` keyword can be used to let the browser do the positioning itself.

The exact positioning is affected by settings for padding, border, margin and the position property.

See also:	Measurement units, `style.bottom`, `style.left`, `style.pixelRight`, `style.posRight`, `style.top`

style.rowSpan (Property)

An indication of how many rows a table cell should span.

Availability:	CSS level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.rowSpan
CSS syntax:	row-span: *aRowCount*	
Argument list:	*aRowCount*	A number of rows to span in a table

This is used when you want to create complex tables, and can be used as an alternative to nesting a table. Avoiding nested tables is good because they can be quite unwieldy.

This is related somewhat to the `rowSpan` property of the `TD` and `TH` objects and controls the same behavior.

See also:	`TABLE` object, `TD.rowSpan`, `TH.rowSpan`

style.rubyAlign (Property)

An MSIE extension to support the alignment of ruby elements.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.rubyAlign
CSS syntax:	ruby-align: *aControl*	
Argument list:	*aControl*	An alignment control keyword

This controls the alignment of the ruby text with respect to the base text.

The following keywords are supported:

- `auto`
- `left`
- `center`
- `right`
- `distribute-letter`
- `distribute-space`
- `line-edge`

See also:	`RUBY` object

style.rubyOverhang (Property)

An MSIE extension to support the alignment of styled elements.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.rubyOverhang`
CSS syntax:	`ruby-overhang: `*`aControl`*	
Argument list:	*`aControl`*	A definition of the kind of overhang to display

This defines the overhang of the ruby text with respect to its base text.

The following keywords are accepted:

- `auto`
- `whitespace`
- `none`

See also:	`RUBY` object

style.rubyPosition (Property)

An MSIE extension to support the alignment of styled elements.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.rubyPosition`
CSS syntax:	`ruby-position: `*`aPosition`*	
Argument list:	*`aPosition`*	A specifier for where the ruby is to be placed

This defines the position of the ruby text with respect to its base text.

The following keywords are accepted:

- `above`
- `inline`

See also:	`RUBY` object

style.scrollbar3dLightColor (Property)

Access to the scrollbar colors.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.scrollbar3dLightColor`
CSS syntax:	`scrollbar-3d-light-color : aColor`	
Argument list:	*`aColor`*	A color value

This provides a means of reading or writing the color value for a scroll box and its scrollbar.

The value is one of the usual color values.

See also:	Color names, Color value

style.scrollbarArrowColor (Property)

Defines the arrow color of a scrollbar.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.scrollbarArrowColor`
CSS syntax:	`scrollbar-arrow-color: aColor`	
Argument list:	*`aColor`*	A color value

This is a color value specified in the normal way.

See also:	Color names, Color value

style.scrollbarBaseColor (Property)

Defines the base color of a scrollbar.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.scrollbarBaseColor`
CSS syntax:	`scrollbar-base-color: aColor`	
Argument list:	*`aColor`*	A color value

This is a color value specified in the normal way.

See also:	Color names, Color value

style.scrollbarDarkShadowColor (Property)

Defines the dark shadow color of a scrollbar.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.scrollbarDarkShadowColor`
CSS syntax:	`scrollbar-dark-shadow-color: aColor`	
Argument list:	*`aColor`*	A color value

This is a color value specified in the normal way.

See also:	Color names, Color value

style.scrollbarFaceColor (Property)

Defines the face color of a scrollbar.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.scrollbarFaceColor`

CSS syntax:	`scrollbar-face-color: aColor`	
Argument list:	*aColor*	A color value

This is a color value specified in the normal way.

See also:	Color names, Color value

style.scrollbarHighlightColor (Property)

Defines the highlight color of a scrollbar

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.scrollbarHighlightColor`
CSS syntax:	`scrollbar-highlight-color: aColor`	
Argument list:	*aColor*	A color value

This is a color value specified in the normal way.

See also:	Color names, Color value

style.scrollbarShadowColor (Property)

Defines the shadow color of a scrollbar.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.scrollbarShadowColor`
CSS syntax:	`scrollbar-shadow-color: aColor`	
Argument list:	*aColor*	A color value

This is a color value specified in the normal way.

See also:	Color names, Color value

style.setAttribute() (Method)

A method for setting attributes in styles.

Availability:	JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myStyle*.setAttribute(*aName*, *aValue*)
Argument list:	*aName*	The name of an attribute to set
	aValue	The value to be stored in the attribute

This method is used to define attribute values that can be accessed or redefined later on.

See also:	style.getAttribute()

style.setExpression() (Method)

An MSIE extension for managing style controls.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myStyle*.setExpression(*aName*, *anExpression*, *aLanguage*)
Argument list:	*aName*	The name of the property
	anExpression	The expression to be evaluated
	aLanguage	The language to use when evaluating the expression

This is used to set the values for expressions within the style handling complex. You can later retrieve the value of these expressions with the `getExpression()` method or delete them with the `removeExpression()` method.

The rules that are used to construct a style are comprised of multiple expressions. You can use this method to assign a new value to an expression within a style item with that value being generated by a callback to a script function.

Given the rule might contain a line such as:

```
width:200px
```

the `setExpression()` method can be applied to the style object containing the rule with that expression like this:

```
myStyle.setExpression("width:", "callBack()", "JavaScript")
```

The value of the width parameter would then be defined by the result of calling the `callBack()` function in the JavaScript context.

The expressions are attached to style properties and can be executed in a variety of languages. The language argument can be one of the following:

- `JavaScript`
- `JScript`
- `VBScript`

See also:	`Element.getExpression(), Element.removeExpression(), Element.setExpression(),style.getExpression(), style.removeExpression()`

Style.size (Property)

Defines the size and orientation of a bounding box on a printed page.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*`.size`
CSS syntax:	`size:` *anAspect*`size:` *aSize anAspect*`size:` *aWidth aHeight anAspect*	
Argument list:	*aSize*	When only one length value is specified, the page is square
	anAspect	An aspect ration control
	aWidth	A page width
	aHeight	A page height

This is part of the printing control mechanism in the CSS level 2 specification.

It provides a way to set the page size and aspect ration of the printed output. This should not affect the display of the page on the screen.

There are three parameter values expected. The first two are numeric values describing the page size. These are expressed in measurement units. You can optionally only include one because the third parameter is a keyword. If you omit both of the numeric values, the page size is assumed to be that of the printer by default. If only one is specified, then a square printable area is defined. Two values control the page width and page height respectively.

Printers may apply a non-printable margin area to the page, and the value you specify here must fit within it otherwise the content may be cropped when it is printed.

The last parameter value is specified using one of the following keywords:

- `auto`
- `portrait`
- `landscape`

See also:	`STYLE.media`, `style.orphans`, `style.page`, `style.pageBreakAfter`, `style.widows`

style.speak (Property)

Part of the aural style control suite that defines whether the content should be spoken out loud or not.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*`.speak`
CSS syntax:	`speak:` *aType*	
Argument list:	*aType*	A speech type

If the browser is equipped to utilize the platform's text to speech capabilities, this will cause the content of the element to be spoken out.

Realistically, the element must have some textual content to speak. The element is not going to make for a very interesting spoken phrase if it is an image, although the `ALT` text of the image might be used if it is defined.

The kind of speech is controlled with these keywords:

- `none`
- `normal`
- `spell-out`

The `none` keyword inhibits the text to speech capabilities.

The `normal` keyword reads the word as best it can. Sometimes this might use phonetic rules to pronounce the word and this may sound strange.

The `spell-out` keyword forces the content to be read out one letter at a time. This may be more helpful if there are complex words that the phonetic rules don't cope with very well, or if the content is a formula or other kind of algebraic expression. This may also be useful for abbreviations or acronyms.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.speakDate (Property)

A format control that dictates the order in which date items are spoken.

Availability:	CSS level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.speakDate
CSS syntax:	speak-date: *aFormat*	
Argument list:	*aFormat*	A date format to speak

Date values may need to be spoken in a manner appropriate to the context of the listener.

This for example may require the order in which the date components should be spoken to manage the regional differences. Typically the American English and UK English language variants exchange the day and month values for example.

The following keywords are supported to select an order for reading out the Day, Month and Year components of a date value:

- `mdy`
- `dmy`
- `ymd`

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.speakHeader (Property)

Part of the aural style control suite that defines whether a table cell's header description is spoken before the content of the cell itself.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.speakHeader
CSS syntax:	speak-header: *aSwitch*	
Argument list:	*aSwitch*	A control for how often to speak table headers

When the contents of a table are being read out, it can become confusing if the table content is simply read out cell by cell. This property indicates how often the column headings should be read out prior to reading the contents of each cell. It supports the keywords `once` and `always`.

The `once` keyword dictates that table column headings should be spoken once at the start of reading out the table.

The `always` keyword mandates that the column heading be spoken immediately prior to each cell in the row.

Other keywords may be added to this property in due course to allow more flexible ways of reading out table headings.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.speakNumeral (Property)

Part of the aural style control suite that defines whether numbers are spoken individually or in a compounded form.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.speakNumeral
CSS syntax:	speak-numeral: *aType*	
Argument list:	*aType*	A manner in which to read numeric values

This property controls how numeric values are read out. Long numeric values would normally be read out one digit at a time, but a year number might be read out as a compound string.

The CSS2 standard dictates that the language for reading the numbers should be defined with the `LANG="..."` HTML tag attribute for the HTML tag that instantiates the object that this style applies to.

The following keywords can be used with this property:

- `none`
- `digits`
- `continuous`

The `none` keyword inhibits the reading out of numbers.

The `digits` keyword forces numeric values to be read out one digit at a time. For example, 2000 would be pronounced as "Two zero zero zero" if the spoken language is English.

The `continuous` keyword forces numeric values to be read out as a word or string of words compounded together. For example, 2000 would be pronounced as "Two thousand" if the spoken language is English.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.speakPunctuation (Property)

Part of the aural style control suite that defines whether punctuation is spoken, or whether it affects the phrasing and delivery of the speech.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	`myStyle.speakPunctuation`
CSS syntax:	`speak-punctuation: aType`	
Argument list:	*`aType`*	A manner in which to speak punctuation characters

For some textual content, it may be important that the punctuation is spelled out one punctuation symbol at a time. In other contexts it may be appropriate for the text-to-speech handler to use the punctuation to control the phasing and intonation of the voice.

The `none` and `code` keywords can be used with this property.

The `none` keyword means that punctuation is interpreted and should affect the phasing of the spoken text.

The `code` keyword forces the text-to-speech output to enumerate each punctuation symbol as it is encountered. How this might affect the inflection and enunciation of the voice may be platform dependant.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.speakTime (Property)

Part of the aural style control suite that defines the format of time values and whether they are spoken in 12 or 24 hour format.

Availability:	CSS level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	`myStyle.speakTime`
CSS syntax:	`speak-time: aFormat`	
Argument list:	*`aFormat`*	A format according to which time values should be spoken

Spoken time values may be controlled with some very minimal keywords to determine whether to use a 12 hour or 24 hour clock for the basis of the spoken time. The following keyword values are appropriate:

- `none`
- `12`
- `24`

The `none` keyword inhibits time speaking, while the other two select an appropriate time pronunciation.

There is scope for more control over the way this property is used in future versions of the CSS standard.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.speechRate (Property)

Part of the aural style control suite that defines the rate at which the text is spoken out loud.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*`myStyle`*`.speechRate`
CSS syntax:	`speech-rate:` *`aSpeed`*	
Argument list:	*`aSpeed`*	A speed at which the text should be spoken

The speech rate may be adjusted to provide a fast talking voice or a slower voice for more complex spoken content.

The property may have a numeric value assigned to indicate how many words should be read per minute. This would be an average value on the grounds that some words may take longer to pronounce than others. The speech rate can be specified as a floating point value for very fine control over the speech rate.

There are also a set of keywords so you can specify a generic absolute value. These are the keywords that are supported:

- `x-slow`
- `slow`
- `medium`
- `fast`
- `x-fast`

Speech rates can be adjusted relative to a parent or containing element object. Assuming a speech rate is defined for a parent object, then you can use the relative speech rate keywords `slower` and `faster`.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, `style.pause`

style.stress (Property)

Part of the aural style control suite that defines the amount of inflection in the voice as items are spoken.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*`.stress`
CSS syntax:	`stress:` *aValue*	
Argument list:	*aValue*	A stress level for the speech

To make the spoken voice sound more interesting (and human-like), this property provides some control over the stress or inflection in the voice as it reads the text contained in the element object.

This property takes a numeric value, either integer or floating point with the value `50` being the normal default. Increasing the stress value of the voice will provide a greater variation in the way that words are enunciated. A value of `zero` would make the voice sound dull and unexciting.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.styleFloat (Property)

A property that allows styled elements to float.

Availability:	JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.styleFloat
CSS syntax:	style-float: *anAlignment*	
Argument list:	*anAlignment*	A floating alignment control value

If you are flowing text around block-structured elements, this property provides some control over which side the block-structured element is aligned on.

The following values can be assigned to this property:

- `none`
- `left`
- `right`

When the property is set to "`none`", the element will appear inline with the text at the position determined by its location in the document source. For small objects this means they will appear to be symbols within the text without casing any line break.

The `style.float` and `style.floatStyle` properties are related.

See also:	`style.float`

style.tableLayout (Property)

Controls how the browser primarily works out table sizing and layout from the content or the sizing HTML tag attributes.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.tableLayout`
CSS syntax:	`table-layout:` *aType*	
Argument list:	*aType*	A kind of table layout to use

When you have a table in a document, it can take some time before the page content first appears. This is because the browser needs to traverse the entire table and load all its content, so that it can work out the width and height of each cell and then sum them together to determine the overall size of the table.

This property provides a switch that enables the browser to use this normal technique or to use an accelerated method, whereby it takes the attributes that are specified with HTML tag attributes and immediately draws the table using those values.

This property accepts the `auto` and `fixed` keywords.

The `auto` keyword selects the normal table sizing and drawing method.

The `fixed` keyword allows the HTML tag attributes controlling the size to be used to draw the table outline, and the widths of the first row of cells are used to set the width of each columns. From there, the table can be lengthened to accommodate any textual content. You should make sure that there is enough size information in the first row of the table to allow it to establish realistic column widths. This means the columns in the first row should have a width equal to or greater than the width of any other cell in the remaining rows for optimum performance. This is less important when the table only contains text, but may be a problem for tables that partially use graphical content.

See also:	`TABLE` object

style.textAlign (Property)

Controls the horizontal alignment of text within the styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.textAlign
CSS syntax:	text-align: *anAlignment*	
Argument list:	*anAlignment*	A style of text alignment to use

Text alignment in the horizontal axis can be controlled with this property. The `style.verticalAlign` property can be used to align objects vertically with respect to their parents.

The following values are appropriate for this property:

- `left`
- `center`
- `right`
- `justify`

Assigning the `center` value is equivalent to placing the `<CENTER>` tags around the text to be center justified. This aligns the text about the center of its width, but leaves the left and right hand sides of the text looking ragged.

The `left` value lines up the left edge of the text but leaves the right side ragged.

The `right` value lines up the right side of the text and leaves the left side ragged.

The `justify` keyword is intended to line up both the left and right sides nice and neatly, using space expansion between words to use up the additional space.

Warnings:

- The `justify` keyword is not defined in the CSS standard but is inconsistently supported across browsers and platforms. In many cases, it may result in a left justified and ragged right appearance rather than the left and right justified effect you would have expected.

See also:	`CAPTION.align`, `JSSTag.textAlign`, `style.textJustify`, `style.verticalAlign`, `style.wordSpacing`

style.textAutospace (Property)

Support for spacing control in ideographic languages used in Asia.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.textAutospace`
CSS syntax:	`text-autospace:` *aControl*	
Argument list:	*aControl*	One of the available keywords

This property accepts the following keywords:

- `none`
- `ideograph-alpha`
- `ideograph-numeric`
- `ideograph-parenthesis`
- `ideograph-space`

style.textDecoration (Property)

Controls decorative additions to the text such as underlines and strike-throughs.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.textDecoration`
CSS syntax:	`text-decoration:` *aStyle*	
Argument list:	*aStyle*	A set of decorative properties for the text

Text decoration is often confused with text style. In word processors you will commonly see italic and bold in the same menu as underline and strike-through. In the context of a web browser they are separated into `fontStyle` and text decoration.

The following decorations can be applied:

- `blink`
- `line-through`
- `overline`
- `underline`
- `none`

You can apply several of these decorations as long as they are space separated in the string being assigned.

To restore text decoration to its default state simply assign an empty string, "". It is also likely that the value "`none`" will cancel the text decoration too, at least on the MSIE browser.

There are individual properties for each of these styles that you can use if you prefer. Those properties are controlled with a Boolean primitive value.

Setting the properties to the value `true` will add that decoration to the text block.

Setting the `style.textDecorationNone` property to `true` will automatically set the others to `false` and cancel the text decoration effect.

See also:	`JSSTag.textDecoration`, `String.blink()`, `String.strike()`, `style.textDecorationBlink`, `style.textDecorationLineThrough`, `style.textDecorationNone`, `style.textDecorationOverline`, `style.textDecorationUnderline`

style.textDecorationBlink (Property)

The blink attribute of the styled element.

Availability:	CSS level – Proposed JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.textDecorationBlink`
CSS syntax:	`text-decoration-blink:` *aSwitch*	
Argument list:	*aSwitch*	A Boolean switch for the text decoration

This is a Boolean value that needs to be set to `true` or `false` to activate the styled appearance.

Warnings:

- The MSIE browser does not support text blinking so even though the property is provided and supported, its effects are ignored.
- This property does not work on some versions of MSIE on the Macintosh platform.

See also:	`String.blink()`,`style.textDecoration`, `style.textDecorationNone`

style.textDecorationLineThrough (Property)

A text decoration style.

Availability:	CSS level – Proposed JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.textDecorationLineThrough`
CSS syntax:	`text-decoration-line-through:` *aSwitch*	
Argument list:	*aSwitch*	A Boolean switch for the text decoration

This is a Boolean value that needs to be set to `true` or `false` to activate the styled appearance.

Warnings:

- This property does not work on some versions of MSIE on the Macintosh platform.

See also:	`String.strike()`,`style.textDecoration`, `style.textDecorationNone`

style.textDecorationNone (Property)

A text decoration style.

Availability:	CSS level – Proposed JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myStyle*.textDecorationNone
CSS syntax:	text-decoration-none: *aSwitch*	
Argument list:	*aSwitch*	A Boolean switch for the text decoration

This is a Boolean value that needs to be set to `true` or `false` to activate the styled appearance.

Warnings:

- This property does not work on some versions of MSIE on the Macintosh platform.

See also:	style.textDecoration, style.textDecorationBlink, style.textDecorationLineThrough, style.textDecorationOverline, style.textDecorationUnderline

style.textDecorationOverline (Property)

A text decoration style.

Availability:	CSS level – Proposed JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	String primitive
JavaScript syntax:	- *myStyle*.textDecorationOverline
CSS syntax:	text-decoration-overline: *aSwitch*
Argument list:	*aSwitch* A Boolean switch for the text decoration

This is a Boolean value that needs to be set to `true` or `false` to activate the styled appearance.

Warnings:

- This property does not work on some versions of MSIE on the Macintosh platform.

See also:	style.textDecoration, style.textDecorationNone

style.textDecorationUnderline (Property)

A text decoration style.

Availability:	CSS level – Proposed JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.textDecorationUnderline`
CSS syntax:	`text-decoration-underline:` *aSwitch*	
Argument list:	*aSwitch*	A Boolean switch for the text decoration

This is a Boolean value that needs to be set to `true` or `false` to activate the styled appearance.

Warnings:

- This property does not work on some versions of MSIE on the Macintosh platform.

See also:	`style.textDecoration`, `style.textDecorationNone`

style.textIndent (Property)

Controls the indentation of text within the styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.textIndent`
CSS syntax:	`text-indent:` *aValue*	
Argument list:	*aValue*	An amount of text indentation

Text blocks can have the first line indented. This might be applied to an element that is instantiated by a `<P>` tag for example.

The value is specified using the normal measurement units such as pixels or floating point fractional em-dash values.

If you want to create a hanging indent effect, the first line can be moved leftwards by specifying a negative value in this property.

Assigning a value of zero to this property restores it to its default condition with all lines being left justified identically.

Warnings:

- Negative indents (sometimes called outdents or hanging indents) may not be supported correctly on all browsers on all platforms. You may need to perform some experiments and platform testing to verify it works on your target user's browsers.
- Some versions of MSIE on Macintosh do not properly support this feature.

See also:	`JSSTag.textIndent,Measurement units`

style.textJustify (Property)

Controls the justification layout of textual content within the styled element.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.textJustify`
CSS syntax:	`text-justify: `*`aControl`*	
Argument list:	*`aControl`*	One of the available control keywords

The text justification can accommodate any of the following keywords:

- `auto`
- `distribute`
- `distribute-all-lines`
- `distribute-center-last`
- `inter-cluster`
- `inter-ideograph`
- `inter-word`
- `kashida`
- `newspaper`

See also:	`style.textAlign`

style.textKashidaSpace (Property)

A means of controlling the Kashida expansion of an Asian font.

Availability:	CSS level – Proposed JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.textKashidaSpace
CSS syntax:	text-kashida-space: *aValue*	
Argument list:	*aValue*	A space scaling value

The value for this property can be specified as a percentage value or with the `inherit` keyword.

style.textShadow (Property)

Controls artistic shadow effects for text in the styled element.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.textShadow
CSS syntax:	text-shadow: *aValueH aValueV aRadius*, ...text-shadow: *aColor aValueH aValueV aRadius*, ...	
Argument list:	*aColor*	A color for the blur
	aValueH	A horizontal blur factor
	aValueV	A vertical blur factor
	aRadius	A blur radius

This is an alternative to using the visual filter effects in MSIE. With this, you can add several shadows to the text, each one having a different color. The shadows can be vertically and horizontally offset from the text on an individual basis. This simulates the effect of the text being lit from several angles with different light sources.

Although the text shadows are layered with the text itself, when the Z order of the text is changed with respect to other elements in the page, the shadows move with it as if they were in some kind of layer group.

The values for the one particular shadow are space separated and should be specified in the correct order since several of them are length values and therefore they share the same namespace.

Each set of arguments for a shadow layer should be separated from the previous one by a comma so there are two levels of delimiting going on.

As an alternative, the keyword value `none` can be assigned to the property to deactivate shadows altogether.

The arguments for each shadow layer are as follows and should be defined in this order:

- A color value (optional and may be omitted).
- The horizontal offset value in spatial coordinates
- The vertical offset value in spatial coordinates
- The blur radius value in spatial coordinates

If the color value is omitted, the color of the element object itself will be used. This value may not have been defined for the element object and may have been inherited from a parent containing element object.

Negative values can be used for the horizontal and vertical offset values.

See also:	Filter – `Shadow()`, Filter object, `style.filter`, `style.zIndex`

style.textTransform (Property)

Controls capitalization of the text within the styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.textTransform`
CSS syntax:	`text-transform:` *aControl*	
Argument list:	*aControl*	A text transformation to apply

Sometimes you may want to format a block of text for a heading. It's likely you will want to modify the capitalization as well as the appearance.

Regardless of the actual upper/lower case combination of letters in the text block, they can be forced to be displayed according to the case rules defined by this attribute.

The following values can be assigned to this property:

- `capitalize`
- `lowercase`
- `uppercase`
- `none`

See also:	`JSSTag.textTransform`

style.textUnderlinePosition (Property)

A means of controlling whether the underline is placed above or below the text.

Availability:	CSS level – Proposed JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*`.textUnderlinePosition`
CSS syntax:	`text-underline-position:` *aControl*	
Argument list:	*aControl*	One of the available keywords

This style property accepts the `above` and `below` keywords.

style.top (Property)

A positioning reference point.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.top`
	-	*myStyle*`.top = `*yCoordinate*
CSS syntax:	`top:` *aValue*	
Argument list:	*aValue*	A positioning value

A CSS-P positioning style attribute that controls the location of an element relative to its containing parent element. The top edges of the two elements are used as the reference points.

The value can be specified in the usual pixel or fractional em-dash measurement units or the `auto` keyword can be used to let the browser do the positioning itself.

The exact positioning is affected by settings for padding, border, margin and the position property.

See also:	Measurement units, `style.bottom`, `style.left`, `style.pixelTop`, `style.posTop`, `style.right`

style.unicodeBidi (Property)

Controls the bi-directionality of Unicode text within the styled element.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*`.unicodeBidi`
CSS syntax:	`unicode-bidi:` *aControl*	
Argument list:	*aControl*	A setting for bi-directional Unicode text

This sets or returns the level of embedding within the bi-directional text drawing mechanism.

The property can accept the following keywords:

- `normal`
- `embed`
- `bidi-override`

style.verticalAlign (Property)

The vertical alignment of the style.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.verticalAlign
CSS syntax:	vertical-align: *anAlignment*	
Argument list:	*anAlignment*	An alignment control value

The `textAlign` property controls the horizontal alignment, this property controls the vertical alignment.

The following values can be assigned to this property to control text alignment within the element's extent rectangle:

- `bottom`
- `top`

These keywords can be assigned to control the way the box is aligned with respect to its surrounding elements in the document content flow:

- `baseline`
- `middle`
- `sub`
- `super`
- `text-bottom`
- `text-top`

You can also define a value with measurement units or as a percentage relative to the next outermost containing parent element.

These alignment values work as you would expect and as are commonly used in the HTML document source.

See also:	`CAPTION.align`, `JSSTag.verticalAlign`, `style.textAlign`

style.visibility (Property)

The visibility of elements in this style.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Opera – 5.0

Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.visibility = *aSwitch*
CSS syntax:	visibility: *aSwitch*	
Argument list:	*aSwitch*	A visibility switch value

This controls the visibility of a DOM or HTML `Element` object. It corresponds to the visibility property of the Netscape Navigator Layer object but uses different keywords.

The following keywords are appropriate for use with this property:

- `inherit`
- `hide`
- `hidden`
- `show`
- `visible`

The `inherit` keyword allows a child element to be visible only when its parent is visible.

If the child element is made explicitly visible or hidden then its visibility will be unaffected by its parent element object. The parent can be invisible, possibly being used for positioning control while the child is on display.

The `hide` and `show` keywords are considered Netscape Navigator 4 and are deprecated. Since `hidden` and `visible` work even for layers, there is no need to ever use `hide` and `show`.

The `hidden` and `visible` keywords mean the same thing as `hide` and `show` but are more portable across browsers and the CSS standard.

See also:	`Layer.visibility`

style.voiceFamily (Property)

Part of the aural style control suite that defines which one of an available set of predefined voices is used to speak the text.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*myStyle*.voiceFamily
CSS syntax:	voice-family: *aFamilyName* ...	
Argument list:	*aFamilyName*	A list of voice family names

Most platforms that support spoken text will provide the facility to speak in a variety of alternative voices. This is somewhat similar to text being displayed on different fonts.

However, the available voices are not likely to be platform independent.

Generic and portable family names have not yet been defined.

You can stack these in the same way that fonts are defined using a left to right precedence rule. The first matching voice family will be used for the spoken text.

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets, `style.fontFamily`

style.volume (Property)

Part of the aural style control suite that defines the dynamic range from soft to loud that the spoken voice will use.

Availability:	CSS level – 2 DOM level – 2	
Property/method value type:	`String primitive`	
JavaScript syntax:	none	*`myStyle`*`.volume`
CSS syntax:	`volume:` *`aValue`*	
Argument list:	*`aValue`*	A volume setting for speech

This controls the loudness of the spoken text. It can be specified with a numeric value which is assumed to be an absolute volume level in the range 0 to 100 and is assumed to be the percentage of the volume range for the platform.

If a percent sign is added, the value inherited from the parent containing element object is multiplied by that percentage for the child element's volume setting.

In addition, these keywords are also supported for defining absolute values:

- `silent`
- `x-soft`
- `soft`
- `medium`
- `loud`
- `x-loud`

Warnings:

- This is not yet supported by any of the browsers.

See also:	Aural style sheets

style.whiteSpace (Property)

Controls how the browser should treat whitespace characters within the document source when it is rendered on the page.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 5.5 Internet Explorer – 5.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.whiteSpace
CSS syntax:	white-space: *aControl*	
Argument list:	*aControl*	A whitespace control value

When the HTML document source is authored, the browser by default will ignore any text formatting within the document apart from that enclosed in <PRE> tags. Between words, multiple spaces are collapsed to single spaces and carriage returns are ignored.

The following keywords can be used to control whitespace handling in the browser:

- `normal`
- `nowrap`
- `pre`

With the `normal` keyword, the browser behaves as it normally would.

The `nowrap` keyword forces the browser to ignore line breaks. There is an implication that whitespace should be preserved, otherwise this setting is no different to the normal keyword.

The `pre` keyword honors all line breaks and whitespace in the text. Setting the `pre` keyword in this property retains the proportional font and simply leaves linebreaks and whitespace intact, unlike text that is enclosed in <PRE> tags, which will be rendered in a monospace font. This means that whitespace used for alignment may still not do what you expect although it may be a way of indenting text. It is not clear from the available documentation whether <BR> tags are honored inside a block styled with the `pre` keyword, but it is likely that they are which is also not the case when enclosing the text inside <PRE> tags. Netscape 6.0 does in fact honor <BR> tags inside an element styled with the `pre` value.

Warnings:

- There are reports that the CSS syntax for this property does not work correctly on Netscape Navigator (even in version 6).
- MSIE 5 meanwhile has problems with the `pre` value when it is applied to this property.

See also:	`JSSTag.whiteSpace`

style.widows (Property)

Defines the minimum number of lines of a paragraph of text that must be visible at the top of a page when a page break is present. This is most likely to occur when printing documents.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.widows
CSS syntax:	windows: *aCount*	
Argument list:	*aCount*	A count of widow lines

Widows and orphans are fragments of text that appear to be formatted incorrectly when a paragraph of text spans a page break.

Windows and orphans are usually controlled together and the usual technique is to specify that an entire paragraph should be kept on the same page. This forces the paragraph to be taken over to the next page in its entirety, even if the flow requires just a single word to be taken over.

The CSS styling controls allow a finer level of control in that you can allow for a paragraph to be split across a page boundary, but specify a lower limit on the number of lines that must be kept on a single page.

This is fine in principle but there can be some contention for the right layout when a very short paragraph is spanning a page break. This will generally be solved simply by forcing the page break to happen before the paragraph causing the whole paragraph to be carried over to the next page.

A widow is that fragment of text that is left at the top of a page when a paragraph encloses a page break. It is the bottom few lines of the paragraph. The integer value in this property controls the minimum number of lines that must be present. This has the effect of moving the page break earlier in the paragraph to satisfy the requirements of the window setting. However, that itself may transgress the setting for the orphan property, leading to the whole paragraph being taken onwards.

See also:	`style.orphans`, `style.pageBreakAfter`, `style.size`

style.width (Property)

The width of a styled element.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.width
CSS syntax:	width: *aWidth*	
Argument list:	*aWidth*	An object width

The object space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

When used to read the width of an object, this returns a value in pixel units with the `px` suffix.

You can also use this property to change the width of a styled element.

The normal range of values specified in measurement units can be used. You can also assign the `auto` keyword to let the browser deduce the width of an object from the document source.

Warnings:

- Note that not all styled elements can be resized by assigning a new value to this property.

See also:	`JSSTag.width`, Measurement units, `style.height`, `style.pixelWidth`, `style.posWidth`

style.wordBreak (Property)

A means of controlling line breaking behavior within words.

Availability:	CSS level – Proposed JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyle*.wordBreak

CSS syntax:	word-break: *aControl*	
Argument list:	*aControl*	One of the available keywords

This is especially useful for controlling line breaking behavior when multiple languages are used.

The following keywords can be used:

- normal
- break-all
- keep-all

style.wordSpacing (Property)

Controls the spacing between words on the page.

Availability:	CSS level – 1 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.01 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myStyle*.wordSpacing
CSS syntax:	word-spacing: *aSwitch*	
Argument list:	*aSwitch*	A control value for word spacing

When the textAlign attribute is set to the justify keyword, this spacing may come into play.

A length value in measurement units can be specified to adjust the spacing between words or the normal keyword can be assigned to let the browser calculate spacing on its own.

Warnings:

- As of MSIE 4.01, this is only available on the Macintosh platform.

See also:	Measurement units, style.textAlign

style.wordWrap (Property)

Controls the word wrapping behavior when the content exceeds the bound of its containing element object.

Availability:	CSS level – Proposed JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myStyle`*`.wordWrap`
CSS syntax:	`word-wrap: `*`aControl`*	
Argument list:	*`aControl`*	One of the available keywords

This property accepts the `normal` and `break-word` keywords.

style.writingMode (Property)

A typographic control for use with Asian fonts.

Availability:	CSS level – Proposed JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.writingMode`
CSS syntax:	`writing-mode: `*`aControl`*	
Argument list:	*`aControl`*	One of the available keywords

This controls the direction that characters are written into the display.

The following keywords are accepted:

- `lr-tb`
- `tb-rl`

style.zIndex (Property)

A value for the position of the element in a Z stacking order.

Availability:	CSS level – 2 DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0 Opera – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyle*.zIndex
	-	*myStyle*.zIndex = *aZValue*
CSS syntax:	z-index: *aZValue*	
Argument list:	*aZValue*	A position in the Z order

Elements can be placed in front of or behind other elements according to the Z index value.

The value should be defined with an integer or can be replaced by the `auto` keyword. The `auto` value is the same as setting this property to zero.

Any elements that share the same `zIndex` value will be stacked in the display using their order as presented in the document source.

Warnings:

- The CSS-P standard name for the property is `z-index` so this property may be accessible with a different name on some implementations.

See also:	`Layer.zIndex`, `style.textShadow`, `style.zOrder`

style.zoom (Property)

Reads and writes a zoom scaling factor for the receiving element object.

Availability:	JScript – 5.5 Internet Explorer – 5.5	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`style.zoom`
CSS syntax:	zoom: *aScaleFactor*	
Argument list:	*aScaleFactor*	A zooming control value

This controls a magnification effect that is applied to an object. The value can be one of the following:

- `normal`
- A floating point multiplier
- A percentage scalar

style.zOrder (Pitfall)

In some documentation, this property is described as an alternative way of controlling the Z ordered location of the styled object.

Warnings:

- So far, no examples of this property have been located. Use the `zIndex` property.

See also:	`style.zIndex`

<STYLE> (HTML Tag)

Style controls can be effected from JavaScript code.

The style mechanisms allow the content and the appearance of the document to be separated. The style sheet can now define the appearance of the page. This also means that styles can be shared and if your server is up to the task, you can serve a different style sheet according to the user agent value defined by the requesting browser.

Currently, CSS1 styling is widely used and CSS2 is gaining acceptance.

As far as we are concerned with JavaScript, we either need to create a style or apply it to part of a document. To create a style in Netscape Navigator 4 we use JSSS (JavaScript Style Sheets), which is functionally equivalent to CSS1. However JSSS is completely unavailable in any other browser, including Netscape Navigator 6.0 and its use for scripts is discouraged.

See also:	`.htc`, `<META>`, `<STYLE TYPE="...">`, CSS, CSS level 1, CSS level 2, `Document.attachEvent()`, `Document.detachEvent()`, `JavaScript Style Sheets`, `Window.attachEvent()`, `Window.detachEvent()`

Cross-references:

Wrox *Instant JavaScript* – page 50

<STYLE TYPE="..."> (HTML Tag Attribute)

The MIME type for a block of JSSS style code.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
Deprecated:	Yes	
HTML syntax:	`<STYLE TYPE="..."> some JSSS Code </STYLE>`	
Argument list:	`...`	A MIME type that signifies JavaScript source text.
	`someCode`	Some script based style text

The same MIME type values that apply to the `<SCRIPT>` tag are also used for the `<STYLE>` tag when this attribute is applied.

When this JavaScript style sheet is evaluated, the interpreter adds the document to the scope chain so that the `tags`, `classes` and `ids` properties of the document object can be accessed directly as if they were global values. This saves accessing them with the `document` object prefix. Thus:

`document.tags` becomes `tags`

`document.classes` becomes `classes`

`document.ids` becomes `ids`

Warnings:

- This functionality is removed from Netscape Navigator 6.0.

See also:	`<META>`, `<SCRIPT TYPE="...">`, `<SCRIPT>`, `<STYLE>`, Adding JavaScript to HTML, JavaScript Style Sheets, MIME types, `SCRIPT.type`, `StyleSheet.type`, `text/JavaScript`

Cross-references:

Wrox *Instant JavaScript* – page 50

StyleSheet object (Object/DOM)

An object that represents a style sheet.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

JavaScript syntax:	-	*myStyleSheet* = *myDocument*.styleSheets[*anIndex*]
	IE	*myStyleSheet* = *myDocument*.createStyleSheet()
Argument list:	*anIndex*	A valid reference to an item in the collection
Object properties:	cssText, disabled, href, id, media, ownerNode, owningElement, owningNode, parentStyleSheet, readOnly, title, type	
Object methods:	addImport(), addRule(), removeRule()	
Collections:	cssRules[], imports[], rules[]	

A style sheet contains many individual style objects which are managed as a collection. These style sheet objects are created by means of the <STYLE> HTML tag or are imported with the <LINK> tag. They can also be created by means of the @import statement inside a style definition.

The Document.styleSheets[] collection contains a reference for every styleSheet object in the document.

Beware that a STYLE object and a style object are different things. A STYLE object is instantiated by the <STYLE> HTML tag and contains properties that reflect its attributes.

This is quite different to a style object which is a member of a styleSheet and describes the rules for a particular style.

DOM level 2 adds the following properties:

- title
- media
- ownerRule

It also adds the following methods:

- insertRule()
- deleteRule()

Warnings:

- Note that MSIE 5 incorrectly names this object class as styleSheet instead of StyleSheet (note the capitalization).

See also:	Document.createStyleSheet(),Document.styleSheets[], Element.style,rule object,Style sheet, StyleSheetList object

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
cssText	-	5.0 +	-	5.0 +	-	-	-
disabled	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	-

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
Href	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	ReadOnly.
id	-	5.0 +	-	5.0 +	-	-	ReadOnly.
media	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	Warning, ReadOnly.
ownerNode	1.5 +	-	6.0 +	-	-	2 +	ReadOnly.
owningElement	-	3.0 +	-	4.0 +	-	-	ReadOnly.
owningNode	-	5.0 +	-	5.0 +	-	-	Warning, ReadOnly.
parentStyleSheet	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	ReadOnly.
readOnly	1.5 +	3.0 +	6.0 +	4.0 +	-	-	ReadOnly.
title	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	Warning, ReadOnly.
type	1.5 +	3.0 +	6.0 +	4.0 +	-	2 +	ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
addImport()	-	3.0 +	-	4.0 +	-	-	-
addRule()	-	3.0 +	-	4.0 +	-	-	-
removeRule()	-	5.0 +	-	5.0 +	-	-	-

StyleSheet.addImport() (Method)

A method for importing to style sheets.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myStyleSheet*.addImport(*aUrl*)
	IE	*myStyleSheet*.addImport(*aUrl*, *anIndex*)
Argument list:	*anIndex*	An index position within the collection where the new style sheet should be inserted
	aUrl	A URL value that points at a CSS documents

This method provides a way to import an external style sheet so its rules become part of the current style sheet.

This is another example of how to build a document hierarchy in a web browser.

If you omit the index location for the style sheet that is being imported, it will be appended to the end of the current collection of style sheets belonging to the document.

See also:	Hierarchy of objects

StyleSheet.addRule() (Method)

A method for adding a rule to a style sheet.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myStyleSheet*.addRule(*aSelector*, *aStyle*)
	IE	*myStyleSheet*.addRule(*aSelector*, *aStyle*, *anIndex*)
Argument list:	*anIndex*	A index in the collection where the new rule should be inserted
	aSelector	The style rule selector string
	aStyle	A series of style settings for this rule

This method allows individual rules to be added to a style sheet. You may want to control the precedence of the rules and so the optional index value can be used to determine where in the collection of rules the new rule is to be inserted.

If the index is omitted, the new rule is simply appended to the style sheet.

You can override rules as long as you are not changing the type of rule being defined. Positional rules must always be overridden by a new positioning rule and cannot be changed to an appearance rule and vice versa.

See also:	Document.createStyleSheet(),Hierarchy of objects, StyleSheet.removeRule(),StyleSheet.rules[]

StyleSheet.cssRules[] (Collection)

A collection of CSS rules belonging to a style sheet. This is another name for the rules[] collection.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myStyleSheet*.cssRules

Property attributes:

ReadOnly.

Refer to:

StyleSheet.rules[]

StyleSheet.cssText (Property)

The CSS text belonging to a style sheet.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myStyleSheet*.cssText

This is the source text of the entire CSS text for the whole style sheet. This is not the same as the `cssText` property for individual style rules. It is equivalent to all of those individual `cssText` values concatenated together.

StyleSheet.disabled (Property)

A property that disables a style sheet.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myStyleSheet*.disabled

This is a Boolean primitive value that enables and disables the rules in a style sheet.

The `DISABLED="..."` HTML tag attribute does not work in all versions of MSIE even though the `styleSheet.disabled` property does.

StyleSheet.href (Property)

The HREF location of a style sheet for download from a server.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myStyleSheet*.href

The path to a document on the server where a style sheet can be loaded from. This will likely correspond to an `HREF="..."` or `SRC="..."` HTML tag attribute in a `<LINK>` HTML tag.

Property attributes:

`ReadOnly`.

StyleSheet.id (Property)

The value of the `ID="..."` HTML tag attribute.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myStyleSheet.id`
HTML syntax:	`<LINK ID="anIDValue"><STYLE ID="anIDValue">`	
Argument list:	*`anIDValue`*	A unique `ID` value within the document

This property is initially defined by the `ID="..."` HTML tag attribute of the `<LINK>` or `<STYLE>` tag that instantiates the `styleSheet` object.

Property attributes:

`ReadOnly`.

StyleSheet.imports[] (Collection)

A collection of all the imported style sheets defined for this style sheet object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`styleSheets object`	
JavaScript syntax:	IE	*`myStyleSheet`*`.imports`

See also:	`StyleSheetList` object, `Collection` object

Property attributes:

`ReadOnly`.

StyleSheet.media (Property)

A description of the target media.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`MediaList object`	
JavaScript syntax:	-	*`myStyleSheet`*`.media`

Warnings:

- DOM level 2 style specification details cover the use of style sheets for several different media. This acts as a selector for one of several alternative appearances depending on the resolution and color capabilities of the target medium the document will be presented on.
- Typically that might be print or video screen.

Property attributes:

`ReadOnly`.

StyleSheet.ownerNode (Property)

The DOM node that owns the style sheet.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`Node object`	
JavaScript syntax:	N	`myStyleSheet.ownerNode`

This is part of the internal node structured hierarchy maintained by the browser as a foundation for its implementation of the DOM model. In many cases this is likely to be the same object as that returned by the `owningElement` property. This property is in preparation for the DOM level 2 styling support.

Note that this property is named incorrectly in MSIE 5 where it is called `owningNode`.

See also:	`StyleSheet.owningNode`

Property attributes:

`ReadOnly`.

StyleSheet.owningElement (Property)

The element that owns the style sheet.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`An object`	
JavaScript syntax:	IE	*`myStyleSheet`*`.owningElement`

This property refers to the owning `LINK` or `STYLE` object that is the parent for the style sheet. From that you can establish a collection of sibling style sheets.

See also:	Hierarchy of objects

Property attributes:

`ReadOnly`.

StyleSheet.owningNode (Property)

The DOM node that owns the style sheet.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Node object`	
JavaScript syntax:	IE	`myStyleSheet.owningNode`

This is part of the internal node structured hierarchy maintained by the MSIE browser as a foundation for its implementation of the DOM model. In many cases this is likely to be the same object as that returned by the `owningElement` property. This property is in preparation for DOM level 2 styling support.

Warnings:

- Note that this property is named incorrectly in MSIE 5 and should be called `ownerNode` according to the DOM level 2 style specification.

See also:	Hierarchy of objects, `StyleSheet.ownerNode`

Property attributes:

`ReadOnly`.

StyleSheet.parentStyleSheet (Property)

The parent stylesheet that styles are cascaded from.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`StyleSheet object`	
JavaScript syntax:	-	*`myStyleSheet`*`.parentStyleSheet`

Where style sheets are imported and if the object reference is a `StyleSheet` object that was the result of an import, then that `StyleSheet` object will have a parent. The value returned by this property is another `StyleSheet` object.

See also:	Hierarchy of objects

Property attributes:

`ReadOnly`.

StyleSheet.readOnly (Property)

The read-only property of a style sheet.

Availability:	JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`myStyleSheet`*`.readOnly`

In the same way that you can lock a rule and make it read-only, you can also use this property to lock an entire style sheet to prevent it from being changed in the browser by JavaScript.

See also:	`rule.readOnly`

Property attributes:

`ReadOnly`.

StyleSheet.removeRule() (Method)

An accessor for removing rules belonging to a style sheet's rules collection.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	`myStyleSheet.removeRule(anIndex)`
Argument list:	*`anIndex`*	The index number of the rule to be removed

If you used the `addRule()` method to create new rules, then this method operates as a complementary tool to remove rules from the style sheet.

See also:	`StyleSheet.addRule()`

StyleSheet.rules[] (Collection)

An array of rules contained within this style sheet.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`SelectorArray` object	
JavaScript syntax:	IE	*`myStyleSheet`*`.rules`

The object class of the rule collection is called a `SelectorArray`.

See also:	`rule` object, `SelectorArray` object, `StyleSheet.addRule()`

Property attributes:

`ReadOnly`.

StyleSheet.title (Property)

This is an advisory title text.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myStyleSheet`*`.title`

Warnings:

- The DOM standard notes that the title value is often specified in the `ownerNode`. That would be the `TITLE="..."` HTML tag attribute for a `<LINK>` tag. It also relates to the pseudo-attribute for the XML style sheet processing instruction.

Property attributes:

`ReadOnly`.

StyleSheet.type (Property)

What sort of style sheet the object represents.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- `myStyleSheet.type`
HTML syntax:	`<STYLE TYPE="...">`

This value returns a string that describes the MIME type for the style sheet document that was used to instantiate the `StyleSheet` object.

Normally, you would see the value "`text/css`" in this property.

See also:	`<SCRIPT TYPE="...">`,`<STYLE TYPE="...">`, `SCRIPT.type`

Property attributes:

`ReadOnly`.

Style sheet (Definition)

A means of abstracting presentation style out of the content in a web page.

The style sheet support is now standardized around the CSS standards and provides a way to make the HTML document relate to the content of the page, while the style sheet contains a definition of how it should look. This has great benefits for the maintenance of a web site and allows you to significantly change the appearance of the whole site simply by specifying a new style sheet.

See also:	`StyleSheet` object

StyleSheetList object (Object/DOM)

An array of style sheet objects provided by MSIE.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myStyleSheets* = *myDocument*.styleSheets
Object properties:	length	
Object methods:	item()	

Warnings:

- MSIE inconsistently names this as a `styleSheets` object rather than a `StyleSheets`, `StyleSheetCollection` or `StyleSheetArray` object.
- DOM level 2 describes this as a `StyleSheetList` which is more consistent. Because this is a collection, DOM allows for the `item()` method to be supported.

See also:	Collection object, Document.styleSheets[], StyleSheet object

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
length	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
item()	1.5 +	5.0 +	6.0 +	5.0 +	-	2 +	-

StyleSheetList.item() (Method)

An accessor for objects in the `StyleSheetList` collection.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	StyleSheet object	
JavaScript syntax:	-	*myStyleSheetList*.item(*anIndex*)
	-	*myStyleSheetList*[*anIndex*]
Argument list:	*anIndex*	The numeric index of the required style sheet

Calling this method with a numeric value that can range from `0` to `1` less than the `length` value selects the required `StyleSheet` object from the collection.

Note that the array like square bracket notation can be used as well.

StyleSheetList.length (Property)

The number of style sheets currently supported by the document.

Availability:	DOM level – 2 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myStyleSheetList*.length

See also:	`Collection.length`

Property attributes:

`ReadOnly`.

SUB object (Object/HTML)

An object that encapsulates the contents of a `<SUB>` tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	<classname relation="parent">Element object</classname>	
JavaScript syntax:	IE	*mySUB* = *myDocument*.all.*anElementID*
	IE	*mySUB* = *myDocument*.all.tags("SUB")[*anIndex*]
	IE	*mySUB* = *myDocument*.all[*aName*]
	-	*mySUB* = *myDocument*.getElementById(*anElementID*)
	-	*mySUB* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*mySUB* = *myDocument*.getElementsByTagName ("SUB")[*anIndex*]
HTML syntax:	`<SUB> ... </SUB>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object

Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDblClick	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyPress	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onKeyUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseDown	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseMove	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOut	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseOver	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onMouseUp	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

Subclasses (Definition)

Subclasses inherit the behavior from their superclass.

When you create a subclass in JavaScript, it is done by building a constructor function. To be completely correct, the constructor should be linked by the constructor property of the prototype property. We overwrite the prototype as an object is created, however the constructor is inherited from the object that was cloned. We need to set the constructor manually. That is done like this:

```
MyNewClass.prototype.constructor = MyNewClass;
```

Warnings:

- There are some versions of Netscape Navigator that set the constructor property to be read-only and this prohibits a script from changing it.
- In Navigator 4, a non-portable low level internal variable can be 'hacked' to fix up the prototype rather than the constructor. This uses the `__proto__` internal property.

See also:	__proto__, Anchor object, Superclasses

SubmitButton object (Object/DOM)

A button in a form that submits the form to the server.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera browser – 3.0	
Inherits from:	<classname relation="parent">Input object</classname>	
JavaScript syntax:	-	*mySubmitButton* = *myDocument*.*aFormName*.*anElementName*
	-	*mySubmitButton* = *myDocument*.*aFormName*.elements[*anItemIndex*]
	IE	*mySubmitButton* = *myDocument*.all.*anElementID*
	IE	*mySubmitButton* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*mySubmitButton* = *myDocument*.all[*aName*]
	-	*mySubmitButton* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*mySubmitButton* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*]
	-	*mySubmitButton* = *myDocument*.getElementById(*anElementID*)
	-	*mySubmitButton* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*mySubmitButton* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
HTML syntax:	<INPUT TYPE="submit">	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The name attribute of an element
	anElementID	The ID attribute of an element
	anItemIndex	A valid reference to an item in the collection
	aFormIndex	A reference to a particular form in the forms collection
Object properties:	type, value	
Object methods:	handleEvent()	
Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onClick, onDblClick, onFilterChange, onFocus, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onRowEnter, onRowExit	

Many properties, methods and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all subclasses of the `Input` object superclass.

There isn't really a `SubmitButton` object class, but it is helpful when trying to understand the wide variety of input element types if we can reduce the complexity by discussing only the properties and methods of a submit button. In actual fact, the object is represented as an item of the `Input` object class.

Event handling support via properties containing function objects was added to `SubmitButton` objects at version 1.1 of JavaScript.

Unlike MSIE, Netscape Navigator does not support the `select()` method or `defaultValue` property for this subclass of the `Input` object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<FORM>
<SELECT ID="IN1">
<OPTION VALUE="-1">Please select an item
<OPTION VALUE="0">Sunday
<OPTION VALUE="1">Monday
<OPTION VALUE="2">Tuesday
<OPTION VALUE="3">Wednesday
<OPTION VALUE="4">Thursday
<OPTION VALUE="5">Friday
<OPTION VALUE="6">Saturday
</SELECT>
<INPUT ID="SUBMIT" TYPE="Submit" VALUE="CLICK ME" onClick="clickMe()">
</FORM>
<SCRIPT>
//MSIE only
function clickMe()
{
   selectedValue  = document.all.IN1.value;

   if(selectedValue == -1)
   {
      alert("You must select an item first!");
   }
   else
   {
      document.all.SUBMIT.click()
   }
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	Element object,Form.elements[],FormElement object,Input object, Input.accessKey,onClick, SubmitButton.handleEvent()

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
type	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly.
value	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
handleEvent()	1.2 +	-	4.0 +	-	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onClick	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
OnKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

Element object, Input object, Node object

SubmitButton.handleEvent() (Method)

Passes an event to the appropriate handler for this object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*mySubmitButton*`.handleEvent(`*anEvent*`)`
Argument list:	*anEvent*	An event to be handled by this object

This applies to Netscape Navigator prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`SubmitButton` object, `Window.handleEvent()`

SubmitButton.type (Property)

The subclass type of this `Input` object.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*mySubmitButton*`.type`

The type value for a `SubmitButton` is always "`submit`". This value is necessary to determine the type of form element, because this object is really an instance of the `Input` class. There is actually no `SubmitButton` class.

See also:	`Input.type`

Property attributes:

`ReadOnly`.

SubmitButton.value (Property)

The text string in the button.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`mySubmitButton`*`.value`

Although this value may be sent back to the web server when the form is submitted, the main purpose of a Submit button is to trigger the form submission. This property provides a convenient means of labelling the button.

Warnings:

- This may be changed in some platforms but not others.

See also:	`Input.value,ResetButton.value`

Subtract (-) (Operator/additive)

Subtracts the right operand from the left operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera browser – 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`anOperand1 - anOperand2`*
Argument list:	*`anOperand1`*	A numeric value to be subtracted from
	`anOperand2`	A numeric value to be subtracted

Subtraction is indicated when two operands are separated by a minus sign.

The value on the right is subtracted from the value on the left. The result is the difference between the two values.

Subtraction behaves identically to addition, as if the formula:

```
a - b
```

had become:

```
a + (-b)
```

The associativity is left to right.

Refer to the operator precedence topic for details of execution order.

See also:	Add (+), Additive operator, Arithmetic operator, Assign value (=), Assignment expression, Associativity, Negation operator (–), Operator Precedence, Subtract then assign (-=)

Cross-references:

ECMA 262 edition 2 – section 11.6.2

ECMA 262 edition 2 – section 11.6.3

ECMA 262 edition 3 – section 11.6.2

Subtract then assign (-=) (Operator/assignment)

Subtracts the right value from the left, modifying the left-hand value.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	`-`	*`anLValue -= anOperand`*
Argument list:	*`anLValue`*	An operand that can be assigned into
	`anOperand`	A value to subtract

Subtracts the right operand from the left operand and assigns the result to the left operand.

This is functionally equivalent to the expression:

```
anOperand1 = anOperand1 - anOperand2;
```

Although this is classified as an assignment operator it is really a compound of an assignment and an additive operator.

The associativity is right to left.

Refer to the operator precedence topic for details of execution order.

The new value of *`anOperand1`* is returned as a result of the expression.

Warnings:

- The operand to the left of the operator must be an LValue. That is, it should be able to take an assignment and store the value.

See also:	Add then assign (+=), Additive operator, Arithmetic operator, Assignment operator, Associativity, Operator Precedence, Subtract (–)

Cross-references:

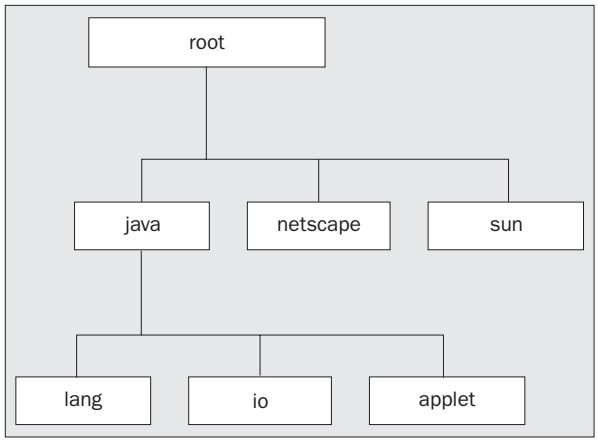
ECMA 262 edition 2 – section 11.13

ECMA 262 edition 3 – section 11.13

sun (Java package)

A short cut reference to the `Packages.sun` object.

Availability:	JavaScript – 1.1 Netscape – 3.0	
Property/method value type:	`JavaPackage sun`	
JavaScript syntax:	N	*myWindow*`.Packages.sun`
	N	*myWindow*`.sun`
	N	`Packages.sun`
	N	`sun`



See also:	`Window.java`, `Window.sun`, `Window.Packages`

Property attributes:

`ReadOnly`.

SUP object (Object/HTML)

An object that encapsulates the contents of a `<SUP>` tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*mySUP* = *myDocument*.all.*anElementID*
	IE	*mySUP* = *myDocument*.all.tags("SUP")[*anIndex*]
	IE	*mySUP* = *myDocument*.all[*aName*]
	-	*mySUP* = *myDocument*.getElementById(*anElementID*)
	-	*mySUP* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*mySUP* = *myDocument*.getElementsByTagName("SUP")[*anIndex*]
HTML syntax:	`<SUP> ... </SUP>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
OnClick	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
OnDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
OnDragStart	-	3.0 +	-	4.0 +	-	-	-	-
OnFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
OnHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
OnKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

Refer to:

`Element` object

super (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section 7.4.3

ECMA 262 edition 3 – section 7.5.3

Superclasses (Definition)

Superclasses are the parent class of subclassed objects.

Superclass mechanisms are used in true object oriented systems to provide a means of generalizing object descriptions and capturing common behaviors into a shared parent class.

A superclass of an object may also have a superclass of its own. Eventually, by walking up the class hierarchy, you should arrive at the topmost class. That is probably the `Object` class but it can be implementation dependent.

See also:	`Anchor` object, Subclasses

switch(...) ... case: ... default: ... (Selector)

Selects one of a set of cases according to a switch value.

Availability:	ECMAScript edition – 3 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server – 3.0	
JavaScript syntax:	-	`switch(`*`aValue`*`) { `*`aCaseTree`*` }`
Argument list:	*`aCaseTree`*	A set of case statements to be selected.
	`aValue`	An integer value used as a selector

The `switch` statement evaluates its expression and selects a labeled statement block for execution according to the value resulting from the expression.

If there is no match, then a `default` case is used.

Each labeled case should be terminated by a `break` keyword to avoid the execution dropping down through into the handler for the next case in the script source. On the other hand, this may be what you intend and so omitting the `break` keyword allows several cases to be matched and handled with a common fragment of code.

The use of `switch` is illustrated in the example. The value enclosed in parentheses is evaluated and its result is used as a selector. It looks for a matching `case` label and executes the code in that block. If it does not match, it uses the `default` block (if there is one).

Unlike the ANSI C version of `switch`, the JavaScript one will match strings as well as integers.

The `break;` statements prevent the code from dropping down into the next block.

The following are all legal case labels:

```
case 0:

case 100*23:

case "abc":

case "aaa" + "bbb":

case Number.NaN:
```

The C language environment dictates that a `switch` statement must be capable of supporting at least 256 individual cases. The ECMAScript standard does not define a limit.

You need to be careful when nesting `switch` statements. The `case` labels will be subordinate to the closest enclosing `switch` given the rules of precedence and block structuring of the code. Case labels must be unique within a single `switch` mechanism, but can duplicate `case` values in other `switch` structures within the same or an enclosing code block.

Warnings:

- Primitive values are allowed as selectors but objects, arrays and functions are not. You could wrap them with `toString()` or `valueOf()` functions though. However the results may be slightly unpredictable unless you thoroughly test the effects of those functions in all your target implementations.
- The value `document.forms.length` is valid for use in a `case` label at JavaScript version 1.2 but may not be later. The values really do have to be constant, and it's possible that a later version of the interpreter may check the `ReadOnly` attribute of any property value that is used in this context.

Example code:

```
// An example switch statement
switch(myValue){
     case 1:      document.write("one");
     break;
     case 'too':
     document.write("two");
     break;
     default:
     document.write("unknown");
     break;
}
```

See also:	`break`, Colon (`:`), `else ...`, Flow control, `if( ... ) ...`, `if( ... ) ... else ...`, Selection statement

Cross-references:

ECMA 262 edition 2 – section 7.4.3

ECMA 262 edition 3 – section 7.5.2

ECMA 262 edition 3 – section 12.11

synchronized (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section 7.4.3

ECMA 262 edition 3 – section 7.5.3

SyntaxError object (Object/core)

A native error object based on the `Error` object.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 Netscape – 6.0	
Inherits from:	`Error object`	
JavaScript syntax:	N	*myError* `= new SyntaxError()`
	N	*myError* `= new SyntaxError(`*aNumber*`)`
	N	*myError* `= new SyntaxError(`*aNumber*`, `*aText*`)`
Argument list:	*aNumber*	An error number
	aText	A text describing the error

This subclass of the `Error` object is used when an exception is caused by a script source text parsing error.

See also:	`catch( ... )`, `Error` object, `EvalError` object, `RangeError` object, `ReferenceError` object, `throw`, `try ... catch ... finally`, `TypeError` object, `URIError` object

Inheritance chain:

`Error` object

Cross-references:

ECMA 262 edition 3 – section 15.1.4.13

ECMA 262 edition 3 – section 15.11.6.4

TABLE object (Object/HTML)

An object that represents a table within a document.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape –6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myTABLE* = *myDocument*.all.*anElementID*
	IE	*myTABLE* = *myDocument*.all.*aTableID*
	IE	*myTABLE* = *myDocument*.all.tags("TABLE")[*anIndex*]
	IE	*myTABLE* = *myDocument*.all[*aName*]
	-	*myTABLE* = *myDocument*.getElementById(*anElementID*)
	-	*myTABLE* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myTABLE* = *myDocument*.getElementsByTagName ("TABLE")[*anIndex*]
HTML syntax:	`<TABLE> ... </TABLE>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align`, `background`, `bgColor`, `border`, `borderColor`, `borderColorDark`, `borderColorLight`, `caption`, `cellPadding`, `cellSpacing`, `cols`, `dataFld`, `dataPageSize`, `dataSrc`, `frame`, `height`, `rules`, `summary`, `tabIndex`, `tFoot`, `tHead`, `width`	

Object methods:	`createCaption()`, `createTFoot()`, `createTHead()`, `deleteCaption()`, `deleteRow()`, `deleteTFoot()`, `deleteTHead()`, `insertRow()`, `nextPage()`, `previousPage()`, `refresh()`
Event handlers:	`onAfterUpdate`, `onBeforeUpdate`, `onBlur`, `onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onFocus`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onResize`, `onRowEnter`, `onRowExit`, `onScroll`, `onSelectStart`
Collections:	`cells[]`, `rows[]`, `tBodies[]`

Tables are a hierarchical means of describing a two dimensional array of cells containing HTML.

Generally speaking the DOM compliant browsers provide a more sophisticated model of the table for access under control of a JavaScript program.

There are a set of related object types that need to be understood to utilize tables most effectively:

- `CAPTION`
- `COL`
- `COLGROUP`
- `TBODY`
- `TD`
- `TFOOT`
- `TH`
- `THEAD`
- `TR`

The following style object properties should also be considered:

- `style.captionSide`
- `style.cellSpacing`
- `style.columnSpan`
- `style.emptyCells`
- `style.rowSpan`
- `style.tableLayout`

See also:	`CAPTION` object, `COL` object, `COLGROUP` object, `Element` object, `Element.offsetParent`, `style.captionSide`, `style.cellSpacing`, `style.columnSpan`, `style.emptyCells`, `style.rowSpan`, `style.tableLayout`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
background	-	3.0 +	-	4.0 +	-	-	-	-
bgColor	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
border	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
borderColor	-	3.0 +	-	4.0 +	-	-	-	-
borderColorDark	-	3.0 +	-	4.0 +	-	-	-	-
borderColorLight	-	3.0 +	-	4.0 +	-	-	-	-
caption	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly
cellPadding	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
cellSpacing	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
cols	-	3.0 +	-	4.0 +	-	-	-	-
dataFld	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
dataPageSize	-	3.0 +	-	4.0 +	-	-	-	-
dataSrc	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
frame	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
height	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	-
rules	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
summary	1.5 +	-	6.0 +	-	-	1 +	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
tFoot	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
tHead	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
createCaption()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
createTFoot()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
createTHead()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
deleteCaption()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
deleteRow()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
deleteTFoot()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
deleteTHead()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
insertRow()	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
nextPage()	-	3.0 +	-	4.0 +	-	-	-	-
previousPage()	-	3.0 +	-	4.0 +	-	-	-	-
refresh()	-	3.0 +	-	4.0 +	-	-	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onFocus	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onResize	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

TABLE.align (Property)

A controlling property for the alignment of table cells within a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTABLE*.align

The horizontal alignment of the `TABLE` object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- `center`
- `left`
- `right`
- `char`
- `justify`

See also:	`TBODY.align`, `TD.align`, `TFOOT.align`, `TH.align`, `THEAD.align`, `TR.align`

TABLE.background (Property)

A URL of an image to be used as the background for a table.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTABLE*`.background`

If a background image is available, then its URL is contained in this property. Changing the value in this property will replace the background with a new one. However, there may be a perceptible delay while the new image is fetched from the web server.

TABLE.bgColor (Property)

The background color value for a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTABLE*`.bgColor`

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	Color names, Color value

TABLE.border (Property)

The width of the border around cells in a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTABLE*.border

The thickness of the border around the table should be specified as an integer value describing a value measured in pixels. This describes the table border drawn with a 3D effect. If you add borders around any elements with the style properties, those describe a different border.

See also:	`TABLE.frame`

TABLE.borderColor (Property)

The color of the border around the cells in a table.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTABLE*.borderColor

You can use this property to determine the color of a border surrounding a table. Note that there are additional properties to determine the highlights and lowlights of the table border coloring.

TABLE.borderColorDark (Property)

The color value of the shadowed edge of the table border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTABLE*.borderColorDark

Table borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the shadowed part of the table border.

TABLE.borderColorLight (Property)

The color value of the highlighted edge of the table border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTABLE*`.borderColorLight`

Table borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the highlighted part of the table border.

TABLE.caption (Property)

The caption text contained in a `CAPTION` object belonging to the `TABLE` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`CAPTION` object	
JavaScript syntax:	-	*myTABLE*`.caption`

This property yields a reference to a `CAPTION` object belonging to the table. That is if you have defined a caption with the `<CAPTION>` HTML tag.

Warnings:

- This property is not fully supported on all versions of the MSIE browser or across all platforms.

See also:	`CAPTION` object

Property attributes:

`ReadOnly`.

TABLE.cellPadding (Property)

The width of the cell padding around the cell contents within a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTABLE*`.cellPadding`

The cell padding describes the space allowed inside the cell but around the cell content. The table grows larger but the border thicknesses stay the same. The growth takes place inside the cells.

TABLE.cells[] (Collection)

A collection of all the cells in a `<TABLE>`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	IE	*myTABLE*`.rows`

This collection includes any cells inside objects instantiated by `<TBODY>`, `<TFOOT>` and `<THEAD>` tags.

This is not necessarily the same as the `cells` collection returned from a `TBODY` object.

Addressing the table by means of its `cells[]` collection allows you to visit all the cells in a single enumeration loop rather than having to nest a loop for each axis of the table.

The columns and rows are different axes of the table. However, they are described in different ways. The `cols` value is simply an integer that is set to the number of columns. The `rows` property is a reference to a collection of objects, one `row` object for each row in the table. Therefore to establish how many cells there are in the table, you need to multiply the `table.cols` value by the `table.rows.length` value.

The `table.cols` value should also be equivalent to `table.rows.cells.length` value as well.

See also:	`Collection` object, `TABLE.cols`, `TABLE.rows[]`, `TBODY.rows[]`, `TFOOT.rows[]`, `THEAD.rows[]`, `TR.rowIndex`

Property attributes:

`ReadOnly`.

TABLE.cellSpacing (Property)

The width of the cell spacing between cells in a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTABLE*`.cellSpacing`

The cell spacing describes the space allowed between adjacent table cells. This moves the cells apart by introducing a few pixels outside the cell border. It has the effect of thickening the border lines between cells.

TABLE.cols (Property)

The number of columns in a table.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myTABLE*`.cols`

The columns and rows are different axes of the table. However, they are described in different ways. The `cols` value is simply an integer that is set to the number of columns. The `rows` property is a reference to a collection of objects, one `row` object for each row in the table. Therefore to establish how many cells there are in the table, you need to multiply the `table.cols` value by the `table.rows.length` value.

The `table.cols` value should also be equivalent to the `table.rows.cells.length` value.

See also:	`style.columnSpan`, `TABLE.cells[]`, `TABLE.rows[]`, `TBODY.rows[]`, `TFOOT.rows[]`, `THEAD.rows[]`, `TR.cells[]`

TABLE.createCaption() (Method)

Creates a new `CAPTION` object for use with the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

Property/method value type:	CAPTION object	
JavaScript syntax:	-	*myTABLE*.createCaption()

You can modify a caption by accessing its innerHTML property in MSIE and Netscape 6.0.

See also:	CAPTION object

TABLE.createTFoot() (Method)

Creates a new TFOOT object for use with the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	TFOOT object	
JavaScript syntax:	-	*myTABLE*.createTFoot()

See also:	TFOOT object

TABLE.createTHead() (Method)

Creates a new THEAD object for use with the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	THEAD object	
JavaScript syntax:	-	*myTABLE*.createTHead()

See also:	THEAD object

TABLE.dataPageSize (Property)

Part of the data-binding model in MSIE that maps table rows to database rows.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myTABLE*`.dataPageSize`

This is used in MSIE when the contents of a table are instantiated from a database fetch. Each row corresponds to a record and the columns correspond to a field in the record structure.

This property determines how many records are assumed to be in each page full of data. That way the table can be traversed one full page at a time.

The `nextPage()` and `previousPage()` methods are used to aid navigation of a table that is paged in this manner.

See also:	`TABLE.nextPage(),TABLE.previousPage(), TABLE.refresh()`

TABLE.deleteCaption() (Method)

Deletes the `CAPTION` object currently belonging to the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTABLE*`.deleteCaption()`

See also:	`CAPTION` object

TABLE.deleteRow() (Method)

Deletes a specified row within the `TABLE` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTABLE*`.deleteRow(`*anIndex*`)`
Argument list:	*anIndex*	The row to delete

See also:	`TR` object

TABLE.deleteTFoot() (Method)

Deletes the table footer from the owning table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTABLE*.deleteTFoot()

See also:	TFOOT object

TABLE.deleteTHead() (Method)

Deletes the table header from the owning table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTABLE*.deleteTHead()

See also:	THEAD object

TABLE.frame (Property)

A control over which of the table cells sides are controlled by the BORDER tag attribute and border property of the TABLE object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTABLE*.frame

This property controls which parts of a table's outer framing border are rendered into the display. The following keywords are supported by this property:

Keyword	Description
above	The top edge only
below	The bottom edge only
border	All four sides
box	Same as border
hsides	Top and bottom edges
lhs	Left edge
rhs	Right edge
void	No sides framed at all
vsides	Left and right sides

The `frame` property affects the outer border around the whole table. You should use the `border` property to control the ruled edges between cells.

See also:	`TABLE.border`, `TABLE.rules`

TABLE.height (Property)

The height of the `TABLE` in pixels.

Availability:	JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTABLE*.height

A table will be sized according to the content within it. The HTML tags for a table allow the table height to be specified as a percentage of the height of the containing HTML element. When the table is rendered, an extent rectangle can be drawn round it. An extent rectangle is the smallest rectangle that will completely enclose the table. This `height` property returns the vertical size of that extent rectangle.

Including height and width information on tables is optional, but it can significantly improve the performance of the layout engine as it renders the web page. This is because the layout engine does not need to wait until the table content has all been fetched before reserving sufficient space for it in the display.

See also:	`TABLE.width`

TABLE.insertRow() (Method)

Insert a new row into the table at a specified row index.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`TR object`	
JavaScript syntax:	-	*myTABLE*`.insertRow(`*anIndex*`)`
Argument list:	*anIndex*	The row at which to insert a new row

See also:	`TR` object

TABLE.nextPage() (Method)

Part of the data binding mechanism that pages the table according to the contents of a database selection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTABLE*`.nextPage()`

When this method is called from your script, the data handler will load the next group of records from the data source and display them in the table cells. The number of records fetched from the database corresponds to the `dataPageSize` property.

See also:	`TABLE.dataPageSize`, `TABLE.previousPage()`, `TABLE.refresh()`

TABLE.previousPage() (Method)

Part of the data binding mechanism that pages the table according to the contents of a database selection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTABLE*`.previousPage()`

When this method is called from your script, the data handler will load the previous group of records from the data source and display them in the table cells. The number of records fetched from the database corresponds to the `dataPageSize` property.

See also:	`TABLE.dataPageSize`, `TABLE.nextPage()`, `TABLE.refresh()`

TABLE.refresh() (Method)

Part of the data binding mechanism that refreshes the table according to the contents of a database selection.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTABLE*.refresh()

When this method is called from your script, the data handler will load the current group of records from the data source and display them in the table cells. The number of records fetched from the database corresponds to the dataPageSize property.

This might be useful if the database is a means of communicating between several users who are accessing the data at the same time. You might set up some means of refreshing automatically with a timeout loop.

See also:	TABLE.dataPageSize, TABLE.nextPage(), TABLE.previousPage()

TABLE.rows[] (Collection)

A collection of all the rows in a <TABLE>.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	Collection object	
JavaScript syntax:	-	*myTABLE*.rows

This collection includes any rows inside objects instantiated by <TBODY>, <TFOOT> and <THEAD> tags.

This is not necessarily the same as the rows collection returned from a TBODY object.

The columns and rows are different axes of the table. However they are described in different ways. The cols value is simply an integer that is set to the number of columns. The rows property is a reference to a collection of objects, one row object for each row in the table. Therefore to establish how many cells there are in the table, you need to multiply the table.cols value by the table.rows.length value.

The table.cols value should also be equivalent to table.rows.cells.length value.

See also:	Collection object, TABLE.cells[], TABLE.cols, TBODY.rows[], TFOOT.rows[], THEAD.rows[], TR.rowIndex

Property attributes:

ReadOnly.

TABLE.rules (Property)

Controls the drawing of border rules around table cells.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTABLE*`.rules`

This property controls which parts of a table's inner borders are rendered into the display. The following keywords are supported by this property:

Keyword	Description
all	All borders are placed around all cells
cols	Borders only delimit columns of cells
groups	Borders are placed between cell groups
none	No borders between cells are displayed at all
rows	Borders only delimit rows

Where borders are used to delimit groups of cells, the groups are established with the `<THEAD>`, `<TBODY>`, `<TFOOT>`, `<COLGROUP>` or `<COL>` HTML tags.

The `rules` property affects the inner borders around the cells in the table. The `frame` property controls the ruled edges around the outside of the table. Table ruling and bordering is quite complex and some time spent messing around with the property values is worthwhile.

Warnings:

- Be careful not to confuse this with the `rules[]` collection of a StyleSheet. These rules are ruled borders not style sheet rules.

See also:	`COL` object, `COLGROUP` object, `TABLE.frame`, `TBODY` object, `TFOOT` object, `THEAD` object

TABLE.summary (Property)

Adds a summary text object to the table.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTABLE*.summary

This is a textual summary of the table for presentation where the table cannot be rendered out. Braille browsers or speaking browsers for the sight impaired would utilize this and it might be useful as a mouse rollover tooltip text.

TABLE.tBodies[] (Collection)

A collection of all `TBODY` objects within a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	-	*myTABLE*.tBodies

If a table has any `<TBODY>` HTML tags defined within its extent, then the `TBODY` objects instantiated by those tags will become members of this collection.

See also:	`TABLE` object, `TBODY` object

Property attributes:

`ReadOnly`.

TABLE.tFoot (Property)

A reference to a `TFOOT` object for the table if there is one defined.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`TFOOT object`	
JavaScript syntax:	-	*myTABLE*.tFoot

If a table has a <TFOOT> HTML tag defined within its extent, then the TFOOT object instantiated by those tags can be accessed via this property. Note that a table can own only one TFOOT object.

See also:	TABLE object, TFOOT object

Property attributes:

ReadOnly.

TABLE.tHead (Property)

A reference to a THEAD object for the table if there is one defined.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	THEAD object	
JavaScript syntax:	-	*myTABLE*.tHead

If a table has a <THEAD> HTML tag defined within its extent, then the THEAD object instantiated by those tags can be accessed via this property. Note that a table can own only one THEAD object.

See also:	TABLE object, THEAD object

Property attributes:

ReadOnly.

TABLE.width (Property)

The width of the TABLE in pixels.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTABLE*.width

A table will be sized according to the content within it. The HTML tags for the table allow the table width to be specified as a percentage of the width of the containing HTML element. When the table is rendered, an extent rectangle can be drawn round it. An extent rectangle is the smallest rectangle that will completely enclose the table. This `width` property returns the horizontal size of that extent rectangle.

Including height and width information on tables is optional but it can significantly improve the performance of the layout engine as it renders the web page. This is because the layout engine does not need to wait until the table content has all been fetched before reserving sufficient space for it in the display.

See also:	`TABLE.height`

TableColElement object (Object/HTML)

A means of accessing cells in a particular column of the table without needing to traverse the rows.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0
Inherits from:	`Element object`
JavaScript syntax:	- *`myTableColElement`* `= new TableColElement()`
Object properties:	`align`, `ch`, `chOff`, `span`, `vAlign`, `width`
Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`

See also:	`COL` object, `COLGROUP` object, `TABLE` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`ch`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`chOff`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`span`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`vAlign`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-
`width`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onHelp	-	5.0 +	-	5.0 +	-	-	-	Warning
onKeyDown	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	5.0 +	6.0 +	5.0 +	-	-	4.0 +	Warning

Inheritance chain

`Element` object, `Node` object

TableColElement.align (Property)

The alignment settings for a column or column group in the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myTableColElement`*`.align`
See also:	`COL` object, `COLGROUP` object, `TABLE` object	

TableColElement.ch (Property)

The alignment character for cells in the column.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myTableColElement`*`.ch`
See also:	`COL` object, `COLGROUP` object, `TABLE` object	

TableColElement.chOff (Property)

The offset of the alignment character.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myTableColElement`*`.chOff`
See also:	`COL` object, `COLGROUP` object, `TABLE` object	

TableColElement.span (Property)

The number of columns that a column group should span.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*`myTableColElement`*`.span`
See also:	`COL` object, `COLGROUP` object, `TABLE` object	

TableColElement.vAlign (Property)

The vertical alignment setting for a column within the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myTableColElement`*`.vAlign`
See also:	`COL` object, `COLGROUP` object, `TABLE` object	

TableColElement.width (Property)

The width of a column within the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTableColElement*.width

See also:	`COL` object, `COLGROUP` object, `TABLE` object

TableSectionElement object (Object/DOM)

DOM level 1 specifies a single object that MSIE implements as `TFOOT` and `THEAD` objects.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myTableSectionElement* = new TableSectionElement()

See also:	`THEAD` object, `TFOOT` object

tags (Property)

An alternative reference to the `document.tags` property in JSS.

Availability:	JavaScript – 1.2 Netscape – 4.0 Deprecated in Netscape 6.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	N	*myDocument*.tags

Warnings:

- This functionality is removed from Netscape 6.0.

See also:	JavaScript Style Sheets, `Document.tags[]`

taint() (Function/global)

A method for controlling secure access to data values.

Availability:	JavaScript – 1.1 Netscape – 3.0 Deprecated

This was removed at version 1.2 of JavaScript. If you encounter it in a script you are maintaining, it is probably wise to seek how it can be removed, otherwise it is likely to cause a run-time error.

Warnings:

- DO NOT USE THIS FUNCTION!

See also:	`Navigator.taintEnabled()`,`untaint()`

TBODY object (Object/HTML)

An object that encapsulates a `<TBODY>` tag within a `<TABLE>` block.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myTBODY* = *myDocument*.all.*anElementID*
	IE	*myTBODY* = *myDocument*.all.tags("TBODY")[*anIndex*]
	IE	*myTBODY* = *myDocument*.all[*aName*]
	-	*myTBODY* = *myDocument*.getElementById(*anElementID*)
	-	*myTBODY* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myTBODY* = *myTable*.tBodies[*anIndex*]
	-	*myTBODY* = *myDocument*.getElementsByTagName ("TBODY")[*anIndex*]
HTML syntax:	`<TBODY> ... </TBODY>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`accessKey`,`align`,`bgColor`,`chOff`,`rows`,`tabIndex`,`vAlign`	

Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`
Collections:	`rows[]`

Each table owns at least one `TBODY` object which is created by default even if you don't enclose some rows within `<TBODY>` tags. Additional `TBODY` objects can be created to break the table into sections if you need to.

The `TBODY` object represents that part of the table which excludes any footer or header cells.

You can access `TBODY` objects by their `ID` HTML tag attribute in the DOM hierarchy or by selecting them from the `tBodies[]` collection belonging to their parent `TABLE` object.

Warnings:

- Note that on some versions of the MSIE browser on the Macintosh platform, you cannot access the `innerHTML`, `innerText`, `outerHTML` or `outerText` properties of a `TBODY` object.

See also:	`Element` object, `Input.accessKey`, `TABLE` object, `TABLE.rules`, `TABLE.tBodies[]`, `TD` object, `TR` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`accessKey`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`bgColor`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`chOff`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`rows`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`tabIndex`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
`vAlign`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning

Table continued on following page

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

TBODY.align (Property)

A control for the alignment of cells within the `TBODY` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTBODY*.align

The alignment of the `TBODY` object with respect to its containing parent object is defined in this property. The available set of alignment specifiers include:

- `center`
- `left`
- `right`
- `char`
- `justify`

See also:	`TABLE.align`, `TD.align`, `TFOOT.align`, `TH.align`, `THEAD.align`, `TR.align`

TBODY.bgColor (Property)

The background color of a `TBODY` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	-	*myTBODY*.bgColor

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	Color names, Color value

TBODY.rows[] (Collection)

A collection of objects, each one containing a description of a row described by a <TR> tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTBODY*.rows

This is not necessarily the same as the rows collection returned from a TABLE object. That is because the rows[] collection belonging to a TBODY object can only list those TR objects that are contained within the <TBODY> tags.

See also:	Collection object, TABLE.cells[], TABLE.cols, TABLE.rows[], TFOOT.rows[], THEAD.rows[], TR.rowIndex

Property attributes:

ReadOnly.

TBODY.vAlign (Property)

A control for the vertical alignment of cells within the TBODY object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTBODY*.vAlign

The vertical alignment of the content within the table cells can be controlled across an entire TBODY extent with this property. The following keywords can be assigned to it:

- baseline
- bottom
- middle
- top

These may be also available on some implementations:

- absbottom
- absmiddle
- baseline
- texttop

See also:	TD.vAlign, TFOOT.vAlign, TH.vAlign, THEAD.vAlign, TR.vAlign

TD object (Object/HTML)

An object that encapsulates a single cell described by a <TD> tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	Element object	
JavaScript syntax:	IE	*myTD* = *myDocument*.all.*anElementID*
	IE	*myTD* = *myDocument*.all.tags("TD")[*anIndex*]
	IE	*myTD* = *myDocument*.all[*aName*]
	-	*myTD* = *myDocument*.getElementById(*anElementID*)
	-	*myTD* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myTD* = *myDocument*.getElementsByTagName ("TD")[*anIndex*]
HTML syntax:	<TD> ... </TD>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	abbr, accessKey, align, axis, background, bgColor, borderColor, borderColorDark, borderColorLight, cellIndex, ch, chOff, colSpan, headers, height, noWrap, rowSpan, scope, tabIndex, vAlign, width	

Event handlers:	onAfterUpdate, onBeforeUpdate, onBlur, onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onSelectStart

This object is instantiated by a <TD> tag that encloses the content of a single data cell.

Some of the property values in this object may be inherited from parent objects such as TABLE, TR and TBODY.

Warnings:

- Note that on some versions of the MSIE browser on the Macintosh platform, you cannot access the innerHTML, innerText, outerHTML or outerText properties of a TD object.

See also:	Element object, Input.accessKey, TABLE object, TBODY object, TH object, TR object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
abbr	1.5 +	-	6.0 +	-	-	1 +	-	-
accessKey	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
axis	1.5 +	-	6.0 +	-	-	1 +	-	-
background	-	3.0 +	-	4.0 +	-	-	-	-
bgColor	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
borderColor	-	3.0 +	-	4.0 +	-	-	-	-
borderColorDark	-	3.0 +	-	4.0 +	-	-	-	-
borderColorLight	-	3.0 +	-	4.0 +	-	-	-	-
cellIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning, ReadOnly
ch	1.5 +	-	6.0 +	-	-	1 +	-	-
chOff	1.5 +	-	6.0 +	-	-	1 +	-	-
colSpan	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
headers	1.5 +	-	6.0 +	-	-	1 +	-	-
height	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
noWrap	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
rowSpan	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
scope	1.5 +	-	6.0 +	-	-	1 +	-	-
tabIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
vAlign	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onResize	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

TD.abbr (Property)

An abbreviation value to be used for header cells in the column where the data cell resides.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTD*.abbr

See also:	`TH.abbr`

TD.align (Property)

A control for the alignment of content within the table cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTD*`.align`

The alignment of the `TD` object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- `center`
- `left`
- `right`
- `char`
- `justify`

See also:	`TABLE.align`, `TBODY.align`, `TFOOT.align`, `TH.align`, `THEAD.align`, `TR.align`

TD.axis (Property)

The names group of related header cells.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTD*`.axis`

See also:	`TH.axis`

TD.background (Property)

A URL for an image to be loaded into the background of the table cell.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTD*`.background`

If a background image is available, then its URL is contained in this property. Changing the value in this property will replace the background with a new one. However there may be a perceptible delay while the new image is fetched from the web server. You might be able to work around that by preloading the image and keeping a reference to it in a variable.

See also:	`TH.background`

TD.bgColor (Property)

The background color for this table cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTD*`.bgColor`

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	Color names, Color value, `TH.bgColor`

TD.borderColor (Property)

The color of the border around this table cell.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTD*`.borderColor`

You can use this property to determine the color of a border surrounding a table data cell. Note that there are additional properties to determine the highlights and lowlights of the table data cell border coloring.

See also:	`TH.borderColor`

TD.borderColorDark (Property)

The color value of the shadowed edge of the table cell border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTD*`.borderColorDark`

Table data cell borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the shadowed part of the table data cell border.

See also:	`TH.borderColorDark`

TD.borderColorLight (Property)

The color value of the highlighted edge of the table cell border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTD*`.borderColorLight`

Table data cell borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the highlighted part of the table data cell border.

See also:	`TH.borderColorLight`

TD.cellIndex (Property)

A zero-based integer value that indicates the position of this cell within the row. This is the horizontal coordinate position of the cell within the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTD*`.cellIndex`

You can access cells by means of the `rows[]` and `cells[]` collections belonging to the `TABLE` and `TR` objects respectively. This property provides the horizontal coordinate to use in the `TR.cells[]` collection to access the object describing this table cell.

Warnings:

- This is not supported on some versions of MSIE running on the Macintosh platform.

See also:	`TH.cellIndex`

Property attributes:

`ReadOnly`.

TD.ch (Property)

The alignment character for cells in a column arrangement.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTD*`.ch`

HTML 4.0 provides for text to be arranged in neat columns within table cells without the need to create additional tables within tables. This method of alignment is selected by setting the `ALIGN="CHAR"` HTML tag attribute. The `CHAR="..."` HTML tag attribute is reflected in this property and is active when the `CHAROFF="..."` HTML tag attribute is present.

See also:	`COL.ch`, `COLGROUP.ch`, `TH.ch`, `THEAD.ch`, `TR.ch`

TD.chOff (Property)

The offset of a column alignment character.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTD*.chOff

The CHAR alignment style requires that an alignment character be specified with the `ch` property and that an offset measured in pixels be defined as its value. The offset value can be defined with the `CHAROFF` HTML tag attribute. The remainder of the string is offset by this distance from the alignment character.

See also:	`COL.chOff`, `COLGROUP.chOff`, `TH.chOff`, `THEAD.chOff`, `TR.chOff`

TD.colSpan (Property)

The number of columns that the table cell spans.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTD*.colSpan

This corresponds to the `COLSPAN` attribute within a `<TD>` HTML tag for a table cell description. It defines how many table columns this column is to span.

Warnings:

- Note that this may affect the value of the `cellIndex` property for subsequent cells within the same row. Although you cannot alter the `cellIndex` value directly, operations that affect the way the table is addressed may move cells around in the collection that represents all cells in the table.

See also:	`COL.span`, `COLGROUP.span`, `style.columnSpan`, `TH.colSpan`

TD.headers (Property)

A list of `ID` attribute values for header cells.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTD*`.headers`

See also:	`TH.headers`

TD.height (Property)

The height in pixels of the table cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTD*`.height`

The table cell space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

See also:	`TH.height`

TD.noWrap (Property)

Controls whether textual content is allowed to wrap within the table cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*myTD*`.noWrap`

This is a Boolean value that controls whether the textual content is wrapped at the right hand window border or not.

If the value `false` is assigned to this property, then words will wrap as the page is drawn. This is good and is the way you would expect a browser to behave. The text will flow according to the space available.

If the value `true` is assigned to this property, the line of text will continue to the right until a `<BR>` or other block level tag is encountered. This will force the horizontal width of the page to be extremely large and the user will need to scroll furiously to be able to see the text and then scroll back again for the start of the next line.

Warnings:

- Only use this if you plan to place line breaks at frequent intervals yourself and really do need to control the line breaks manually. The effect is as if you had placed `<PRE>` tags around the content of the cell, although the font may still be a proportional font rather than the monospaced font that `<PRE>` sets up by default.

See also:	`TH.noWrap`

TD.rowSpan (Property)

Indicates how many rows the table cell is intended to span.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`Number primitive`
JavaScript syntax:	- *myTD*`.rowSpan`

This is used when you want to create complex tables. This technique may be an alternative to nesting a table. That is always good because sometimes nested tables can become very unwieldy.

See also:	`style.rowSpan`, `TH.rowSpan`

TD.scope (Property)

The scope covered by header cells.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0

Property/method value type:	String primitive	
JavaScript syntax:	N	*myTD*.scope

See also:	TH.scope

TD.vAlign (Property)

The vertical alignment of content within the table cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTD*.vAlign

The vertical alignment of the content within the table cells can be controlled for a single table cell with this property. The following keywords can be assigned to it:

- baseline
- bottom
- middle
- top

These may be also available on some implementations:

- absbottom
- absmiddle
- baseline
- texttop

This value will override the setting for a row or TBODY extent.

See also:	TBODY.vAlign, TFOOT.vAlign, TH.vAlign, THEAD.vAlign, TR.vAlign

TD.width (Property)

The width in pixels of the table cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myTD*`.width`

The table cell space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

See also:	`TH.width`

telnet: URL (Request method)

Opens up a telnet client to do terminal mode access.

Open a telnet application and connect to the telnet port on the target server. You should be asked for a username and password. This will then give you access to a command line interface on the target host as long as you are validated.

See also:	`javascript:` URL, `mailbox:` URL, Security policy, URL

Ternary operator (Definition)

An operator that requires three arguments.

There is only one ternary operator in JavaScript. It is the `?:` conditional execution operator.

See also:	Binary operator, Conditionally execute (?:), Unary operator

Cross-references:

Wrox *Instant JavaScript* – page – 18

Wrox *Instant JavaScript* – page – 21

Text object (Object/DOM)

The DOM level 1 specification describes a `Text` object which MSIE and Netscape 6.0 implement as a `textNode` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
JavaScript syntax:	-	*myText* = new Text()

Refer to:

`textNode` object

text/JavaScript (MIME type)

A MIME type that indicates the content is a JavaScript source text.

This is otherwise known as `application/x-javascript`, although `text/JavaScript` should be used.

See also:	`<SCRIPT TYPE="...">`, `<STYLE TYPE="...">`, MIME types

TEXTAREA object (Object/DOM)

A multiple line text cell in a form.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Inherits from:	`Input` object	
JavaScript syntax:	IE	*myTEXTAREA* = *myDocument*.all.*anElementID*
	IE	*myTEXTAREA* = *myDocument*.all.tags ("TEXTAREA")[*anIndex*]
	IE	*myTEXTAREA* = *myDocument*.all[*aName*]
	-	*myTEXTAREA* = *myDocument*.getElementById(*anElementID*)
	-	*myTEXTAREA* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myTEXTAREA* = *myDocument*.getElementsByTagName ("TEXTAREA")[*anIndex*]

HTML syntax:	`<TEXTAREA> ... </TEXTAREA>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`cols`, `readOnly`, `rows`, `type`, `value`, `wrap`	
Object methods:	`handleEvent()`, `select()`	
Event handlers:	`onAfterUpdate`, `onBeforeUnload`, `onBeforeUpdate`, `onBlur`, `onChange`, `onDragStart`, `onErrorUpdate`, `onFilterChange`, `onFocus`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onRowEnter`, `onRowEnter?`, `onRowExit`, `onScroll`, `onSelect`, `onSelectStart`	

Many properties, methods and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

Untypically, there actually is a `TEXTAREA` class supported by MSIE. Most other kinds of input are simply an instance of the `Input` object class. Netscape prior to version 6.0 internally represents this object as a sub-class of the `Input` object even though it is created by a different HTML tag.

Event handling support via properties containing function objects was added to `TEXTAREA` objects at version 1.1 of JavaScript, but this will have changed to reflect the new DOM event model for Netscape 6.0.

Unlike MSIE, the Netscape 4 implementation of this sub-class of the `Input` object does not support the `click()` method or the `onSelect` event.

The example below seems to be supported by all browsers apart from Netscape 6.0. On this browser, the escape sequence `\x0D` needs to be changed to `\n` in order for the example to work.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<FORM>
Type some lines of text into the text area, click the button and they will be
sorted.<BR><BR>
<TEXTAREA VALUE="" NAME="BOX_A" ROWS=15 COLS=39></TEXTAREA><BR>
<INPUT TYPE="button" VALUE="Reveal" onClick="handleClick()">
</FORM>
<SCRIPT>
function handleClick()
{
   myString = document.forms[0].BOX_A.value;
   myArray = myString.split("\x0d");
   myArray.sort();
   document.forms[0].BOX_A.value = myArray.join("\x0d");
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Element` object, `Element.isTextEdit`, `Form.elements[]`, `Input` object, `Input.accessKey`, `onChange`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `TEXTAREA.handleEvent()`, `TextRange` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`cols`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`readOnly`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
`rows`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`wrap`	-	3.0 +	-	4.0 +	-	-	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-
`select()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUnload`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onChange`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onErrorUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onRowEnter`	-	3.0 +	-	4.0 +	-	-	-	-
`onRowEnter?`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`onRowExit`	-	3.0 +	-	4.0 +	-	-	-	-
`onScroll`	-	3.0 +	-	4.0 +	-	-	-	-
`onSelect`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

TEXTAREA.cols (Property)

The number of columns that a `TextArea` may contain.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTEXTAREA*`.cols`

The text area in a form is sized by columns and rows in the default font used for input items. This property reflects the `COLS` HTML tag attribute in the `<TEXTAREA>` HTML tag that instantiates this object.

Style sheets are about the only way you can control the appearance of these cells because it is the only way the default input font can be changed. If the font size changes then the size of the `TEXTAREA` extent rectangle must change too. This might affect the layout of the page.

TEXTAREA.handleEvent() (Method)

Passes an event to the appropriate handler for this object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myTEXTAREA*`.handleEvent(`*anEvent*`)`
Argument list:	*anEvent*	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`TEXTAREA` object, `Window.handleEvent()`

TEXTAREA.readOnly (Property)

When set to `true`, the `TextArea` cannot be modified.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Boolean value`	
JavaScript syntax:	-	*myTEXTAREA*`.readOnly`

The `TextArea` content is defined but cannot be changed by the user. You may want to control this property based on JavaScript code. If the property is set `true` from the HTML document source, you will have to unlock the field before the user can modify the field content.

See also:	`Input.readOnly`, `Password.readOnly`, `TextCell.readOnly`

Property attributes:

`ReadOnly`.

TEXTAREA.rows (Property)

The number of rows in a text area.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTEXTAREA*`.rows`

The text area in a form is sized by columns and rows in the default font used for input items. This property reflects the `ROWS` HTML tag attribute in the `<TEXTAREA>` HTML tag that instantiates this object.

Style sheets are about the only practical way you can control the appearance of these cells because it is the only way the default input font can be changed. If the font size changes then the size of the `TEXTAREA` extent rectangle must change too. This might affect the layout of the page.

TEXTAREA.select() (Method)

Selects all the text within the <TEXTAREA> cell allowing it to be cut and pasted by the user.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera browser – 3.0	
JavaScript syntax:	-	*myTEXTAREA*.select()

If the browser supports a Selection object or TextRange objects, you may then be able to access the selected text using JavaScript. Of course in a form object, the text of the whole object can also be accessed, but this may not be what was selected because the user may select all or part of a page, and that selection may span several form elements or be only part of a form element.

See also:	Input.select()

TEXTAREA.type (Property)

The type indicator for this Input object.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTEXTAREA*.type

This property is supported to remain consistent with all other form elements. The value returned by a TEXTAREA object should always be "textarea".

This allows you to determine the input type when the form elements array is traversed. Then you can select appropriate handling based on the type value for the element.

See also:	Input.type

Property attributes:

ReadOnly.

TEXTAREA.value (Property)

The text string that has been entered into the text area.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTEXTAREA*`.value`

This may be the same as the `defaultValue` if the user has not yet entered any text into the `TEXTAREA`.

See also:	`Input.value`, `TextCell.value`

TEXTAREA.wrap (Property)

Controls the kind of word wrapping effect within the text area.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTEXTAREA*`.wrap`

You would normally want the text to wrap within a text area field so that you could see it all. However there may be times when other text wrapping regimes could be useful.

This property will accept the following keywords:

- `off`
- `physical`
- `virtual`

The `off` keyword switches word wrapping off altogether.

The `physical` keyword wraps text as it reaches the right border. It will also send the carriage returns back to the server when the form data is submitted.

The `virtual` keyword shows the word wrapping effect on the screen but the carriage returns are not sent back to the server which remains unaware that any wrapping has occurred. This is probably the most useful setting.

TextCell object (Object/DOM)

A single line text cell in a form.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Inherits from:	`Input object`	
JavaScript syntax:	-	*myTextCell* = *myDocument*.*aFormName*.*anElementName*
	-	*myTextCell* = *myDocument*.*aFormName*.elements[*anItemIndex*]
	IE	*myTextCell* = *myDocument*.all.*anElementID*
	IE	*myTextCell* = *myDocument*.all.tags("INPUT")[*anIndex*]
	IE	*myTextCell* = *myDocument*.all[*aName*]
	-	*myTextCell* = *myDocument*.forms[*aFormIndex*].*anElementName*
	-	*myTextCell* = *myDocument*.forms[*aFormIndex*].elements[*anItemIndex*]
	-	*myTextCell* = *myDocument*.getElementById(*anElementID*)
	-	*myTextCell* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myTextCell* = *myDocument*.getElementsByTagName("INPUT")[*anIndex*]
HTML syntax:	`<INPUT TYPE="text">`	
Argument list:	*anIndex*	A valid reference to an item in the collection
	aName	The `NAME` attribute of an element
	anElementID	The `ID` attribute of an element
	anItemIndex	A valid reference to an item in the collection
	aFormIndex	A reference to a particular form in the `forms` collection
Object properties:	`maxLength`, `readOnly`, `size`, `type`, `value`	
Object methods:	`handleEvent()`, `select()`	
Event handlers:	`onAfterUpdate`, `onBeforeUpdate`, `onBlur`, `onChange`, `onFilterChange`, `onFocus`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onRowEnter`, `onRowExit`, `onSelect`	

Many properties, methods and event handlers are inherited from the `Input` object class. Refer to topics grouped with the "Input" prefix for details of common functionality across all sub-classes of the `Input` object super-class.

There isn't really a `TextCell` object class, but it is helpful when trying to understand the wide variety of input element types if we can reduce the complexity by discussing only the properties and methods of a text cell. In actual fact, the object is represented as an item of the `Input` object class.

Event handling support via properties containing function objects was added to `TextCell` objects at version 1.1 of JavaScript.

The Netscape implementation of this sub-class of the `Input` object does not support as wide a variety of events as the MSIE implementation. In particular, the keyboard events are not supported.

The WebTV set-top box does not support the `onkeypress` event handler for this object type prior to the Summer 2000 release.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<DIV ID="RESULT">?</DIV>
<FORM>
<INPUT TYPE="text" VALUE="" NAME="BOX_A"><BR>
<INPUT TYPE="text" VALUE="" NAME="BOX_B"><BR>
<INPUT TYPE="text" VALUE="" NAME="BOX_C"><BR>
<INPUT TYPE="text" VALUE="" NAME="BOX_D"><BR><BR>
<INPUT TYPE="button" VALUE="Reveal" onClick="handleClick()">
</FORM>
<SCRIPT>
function handleClick()
{
   myString  = "[";
   myString += document.forms[0].elements.BOX_A.value;
   myString += "] [";
   myString += document.forms[0].elements.BOX_B.value;
   myString += "] [";
   myString += document.forms[0].elements.BOX_C.value;
   myString += "] [";
   myString += document.forms[0].elements.BOX_D.value;
   myString += "]";
   document.all.RESULT.innerText = myString;
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Element` object, `Element.isTextEdit`, `Form.elements[]`, `FormElement` object, `Input` object, `Input.accessKey`, JellyScript, `onChange`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `TextCell.handleEvent()`, `TextRange` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`maxLength`	-	3.0 +	-	4.0 +	-	-	-	-
`readOnly`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly
`size`	-	3.0 +	-	4.0 +	-	-	-	Warning
`type`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	ReadOnly
`value`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`handleEvent()`	1.2 +	-	4.0 +	-	-	-	-	-
`select()`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onAfterUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBeforeUpdate`	-	3.0 +	-	4.0 +	-	-	-	-
`onBlur`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
`onChange`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onFocus`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onRowEnter`	-	3.0 +	-	4.0 +	-	-	-	-
`onRowExit`	-	3.0 +	-	4.0 +	-	-	-	-
`onSelect`	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	-

Inheritance chain:

`Element` object, `Input` object, `Node` object

TextCell.handleEvent() (Method)

Pass an event to the appropriate handler for this object.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*`myTextCell`*`.handleEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An event to be handled by this object

This applies to Netscape prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 3 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an `Event` object that contains information about the event.

See also:	`TextCell` object, `Window.handleEvent()`

TextCell.maxLength (Property)

The maximum length of values to be entered into the text cell.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myTextCell*`.maxLength`

This defines the maximum number of characters that are allowed to be entered into the text field. The browsers differ in how they handle this value. Some will warn the user with a beep or flash on the screen, others simply stop accepting keystrokes when this number of characters have been entered.

See also:	`Input.maxLength`, `Password.maxLength`

TextCell.readOnly (Property)

When set to `true`, the text cell contents cannot be changed.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextCell*`.readOnly`

The `TextCell` content is defined but cannot be changed by the user. You may want to control this property based on JavaScript code. If the property is set `true` from the HTML document source, you will have to unlock the field before the user can modify the field content.

See also:	`Input.disabled`, `Input.readOnly`, `Password.readOnly`, `TEXTAREA.readOnly`

Property attributes:

`ReadOnly`.

TextCell.select() (Method)

Selects all the textual content of a `<INPUT>` text cell allowing the user to cut and paste it if they so wish.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
JavaScript syntax:	-	*myTextCell*.select()

If the browser supports a `Selection` object or `TextRange` objects, you may then be able to access the selected text using JavaScript. Of course in a `form` object, the text of the whole object can also be accessed, but this may not be what was selected because the user may select all or part of a page and that selection may span several form elements or only part of a form element.

See also:	`Input.select()`

TextCell.size (Property)

The number of characters that have been typed into the text cell.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextCell*.size

This is an approximate measure at best. You cannot be sure how wide this box really needs to be when using a proportionally spaced font in it. The browser will size the box close to an optimal size to cope with the specified number of characters.

Warnings:

- It can be quite distracting if the box size is too small to accommodate the `maxLength` number of characters. This can leave the user having to do some cumbersome select actions with the mouse or use arrow keys to reveal the hidden parts of the textual content of the box.

See also:	`Input.size`, `Password.size`

TextCell.type (Property)

The type of this `Input` object.

Availability:	DOM level – 1 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTextCell*.type

The value of this property for a `TextCell` object must always be "`text`".

This value is necessary to determine the type of form element because this object is really an instance of the `Input` class and not the `TextCell` class. There is actually no `TextCell` class.

See also:	`Input.type`

Property attributes:

`ReadOnly`.

TextCell.value (Property)

The text string that has been entered into the text cell.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTextCell*.value

This may be the same as the `defaultValue` if the user has not yet entered any text into the `TextCell`.

See also:	`Input.value`, `TEXTAREA.value`

textNode object (Object/DOM)

A string of text represented as a node within the document hierarchy.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Inherits from:	`CharacterData object`	
JavaScript syntax:	IE	*myTextNode* = *myDocument*.all.tags("TEXT")[*anIndex*]
	-	*myTextNode* = *myDocument*.createTextNode(*someData*)
Argument list:	*someData*	Textual content for the text node
	anIndex	A selector within a collection of text nodes
Object properties:	`data, length`	
Object methods:	`splitText()`	

The MSIE browser models the document as a collection of nodes. Clearly, an HTML tag corresponds to an object. However, what isn't so obvious is that the text in between HTML tags is collected together and represented by a `textNode` object.

These `textNodes` are generally accessible as child objects belonging to an object instantiated by an HTML tag.

For example:

```
AAA<P>BBB<P>CCC
```

can be accessed as follows:

The `<P>` tags are objects which are members of the `document.getElementsByTagName("P")` collection. The text "BBB" is referenced through the `firstChild` property of the `P` object instantiated by the first `<P>` tag. The text "CCC" is a `textNode` object referenced via the `firstChild` property of the second `<P>` tag.

The DOM level 3 specification is expected to add the following method to the `textNode` object:

- `isWhitespaceInElementContent()`

See also:	`Attribute.nodeName, Document.createTextNode()`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`data`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-
`length`	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
splitText()	1.5 +	5.0 +	6.0 +	5.0 +	-	1 +	-

Inheritance chain:

CharacterData object, Node object

textNode.data (Property)

The data associated with a text node within the document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTextNode*.data

A text node is that textual content in between objects that represent HTML tags. If the HTML tags are bricks in a wall, then the text is the mortar joining them all together. The text nodes could also be called interstitial nodes because they exist in the cracks between HTML Elements.

textNode.length (Property)

The length of the text in a textNode object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	Number primitive	
JavaScript syntax:	-	*myTextNode*.length

The text enclosed in a text node can be measured for length by this property. However it is still stored as a string value in the data property and you should not assume that it is an array simply because it has a length property.

Property attributes:

ReadOnly.

textNode.splitText() (Method)

A means of splitting the text in a text node into two so that HTML can be placed in between them.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0	
Property/method value type:	`textNode object`	
JavaScript syntax:	-	*myTextNode*`.splitText(`*anOffset*`)`
Argument list:	*anOffset*	The location within the text data to split the text at

This is functionally similar to splitting a `String` object. The character index, using a zero-based counting scheme where the split is to occur, should be passed as an argument.

The method returns the text to the right of the split. The receiving object is truncated at the split point. You can then append some HTML to the receiving object and then append the text object that was returned by the method.

TextRange object (Object/JScript)

An object that represents part of the text stream of an HTML document.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myTextRange* `=` *myElement*`.createTextRange()`
Object properties:	`boundingHeight,boundingLeft,boundingTop, boundingWidth,htmlText,text`	
Object methods:	`collapse(),compareEndPoints(),duplicate(), execCommand(),expand(),findText(),getBookmark(), getBoundingClientRect(),getClientRects(), inRange(),isEqual(),move(),moveEnd(),moveStart(), moveToBookmark(),moveToElementText(), moveToPoint(),parentElement(),pasteHTML(), queryCommandEnabled(),queryCommandIndeterm(), queryCommandState(),queryCommandSupported(), queryCommandText(),queryCommandValue(),select(), setEndPoint()`	

The main purpose of a `TextRange` object is to encapsulate that part of the document text that depends on the user having used the mouse to select a portion of text. A `TextRange` object can also encapsulate an insertion point in the text of a document.

An insertion point is encapsulated by creating a TextRange object that is of zero length.

A new TextRange object is instantiated with the createTextRange() method which can be applied to the following object types which can contain selectable text:

- BODY
- TextCell
- TEXTAREA

The TextCell and TEXTAREA objects are members of the Form Element category, sometimes called Input Elements.

When you have created a TextRange object, you can then manipulate its properties to select just that portion of text in the document that you want.

Once you have marked the text you want within the start and end points, you can replace the content of the TextRange with the pasteHTML() method to operate on the text in the document. However, there may be limitations on when and how you can do this depending on the extent of the TextRange object's boundaries.

Warnings:

- There are platform limitations to the TextRange object that mean it is only functional on the Windows platform within MSIE. This limits the audience for your ingenuity in using it, but within a captive environment, this capability can still be useful.

See also:	BODY.createTextRange(), Button object, BUTTON object, Element.isTextEdit, Element.parentTextEdit, Input.createTextRange(), Selection object, selection.createRange(), TEXTAREA object, TextCell object

Property	JavaScript	JScript	N	IE	Opera	Notes
boundingHeight	-	3.0 +	-	4.0 +	-	ReadOnly
boundingLeft	-	3.0 +	-	4.0 +	-	ReadOnly
boundingTop	-	3.0 +	-	4.0 +	-	ReadOnly
boundingWidth	-	3.0 +	-	4.0 +	-	ReadOnly
htmlText	-	3.0 +	-	4.0 +	-	ReadOnly
text	-	3.0 +	-	4.0 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
collapse()	-	3.0 +	-	4.0 +	-	-
compareEndPoints()	-	3.0 +	-	4.0 +	-	-
duplicate()	-	3.0 +	-	4.0 +	-	-
execCommand()	-	3.0 +	-	4.0 +	-	-

Table continued on following page

Method	JavaScript	JScript	N	IE	Opera	Notes
`expand()`	-	3.0 +	-	4.0 +	-	-
`findText()`	-	3.0 +	-	4.0 +	-	-
`getBookmark()`	-	3.0 +	-	4.0 +	-	-
`getBoundingClientRect()`	-	5.0 +	-	5.0 +	-	-
`getClientRects()`	-	5.0 +	-	5.0 +	-	-
`inRange()`	-	3.0 +	-	4.0 +	-	-
`isEqual()`	-	3.0 +	-	4.0 +	-	-
`move()`	-	3.0 +	-	4.0 +	-	-
`moveEnd()`	-	3.0 +	-	4.0 +	-	Warning
`moveStart()`	-	3.0 +	-	4.0 +	-	Warning
`moveToBookmark()`	-	3.0 +	-	4.0 +	-	-
`moveToElementText()`	-	3.0 +	-	4.0 +	-	-
`moveToPoint()`	-	3.0 +	-	4.0 +	-	-
`parentElement()`	-	3.0 +	-	4.0 +	-	-
`pasteHTML()`	-	3.0 +	-	4.0 +	-	-
`queryCommandEnabled()`	-	3.0 +	-	4.0 +	-	-
`queryCommandIndeterm()`	-	3.0 +	-	4.0 +	-	-
`queryCommandState()`	-	3.0 +	-	4.0 +	-	-
`queryCommandSupported()`	-	3.0 +	-	4.0 +	-	-
`queryCommandText()`	-	3.0 +	-	4.0 +	-	-
`queryCommandValue()`	-	3.0 +	-	4.0 +	-	-
`select()`	-	3.0 +	-	4.0 +	-	-
`setEndPoint()`	-	3.0 +	-	4.0 +	-	-

Inheritance chain:

`Element` object, `Node` object

TextRange.boundingHeight (Property)

The height of the extent rectangle around selected text on the page.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myTextRange`*`.boundingHeight`

`TextRange` objects use the start and end points to map into the physical display of the text on the screen. Although it may not be visible, the text range nevertheless corresponds to a spatial extent rectangle that can be described using pixel coordinates. The value of this property is the height of the extent rectangle that currently encloses the text encapsulated by the `TextRange` object.

See also:	`TextRange.boundingWidth`

Property attributes:

`ReadOnly`.

TextRange.boundingLeft (Property)

The left edge of a selected text on the page.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myTextRange`*`.boundingLeft`

`TextRange` objects use their start and end points to map into the physical display of the text on the screen. Although it may not be visible, the text range nevertheless corresponds to a spatial extent rectangle that can be described using pixel coordinates. This property's value is the X coordinate of the left edge of the extent rectangle that currently encloses the text encapsulated by the `TextRange` object.

Property attributes:

`ReadOnly`.

TextRange.boundingTop (Property)

The top edge of a selected text on the page.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myTextRange`*`.boundingTop`

`TextRange` objects use their start and end points to map into the physical display of the text on the screen. Although it may not be visible, the text range nevertheless corresponds to a spatial extent rectangle that can be described using pixel coordinates. This property's value is the Y coordinate of the top edge of the extent rectangle that currently encloses the text encapsulated by the `TextRange` object.

Property attributes:

`ReadOnly`.

TextRange.boundingWidth (Property)

The width of a selected text on the page.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myTextRange`*`.boundingWidth`

`TextRange` objects use their start and end points to map into the physical display of the text on the screen. Although it may not be visible, the text range nevertheless corresponds to a spatial extent rectangle that can be described using pixel coordinates. The value of this property is the width of the extent rectangle that currently encloses the text encapsulated by the `TextRange` object.

See also:	`TextRange.boundingHeight`

Property attributes:

`ReadOnly`.

TextRange.collapse() (Method)

A method that shrinks a text range to an insertion point.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*`myTextRange`*`.collapse()`
	IE	*`myTextRange`*`.collapse(`*`aFlag`*`)`
Argument list:	*`aFlag`*	A Boolean value to indicate which end of the text range should become the insertion point

You can call this method to shrink the `TextRange` to an insertion point either at the start or end of the text range.

If no value is passed as an argument, then the insertion point is taken from the start of the text range.

A Boolean `true` value also uses the startpoint as the new insertion point.

A Boolean value of `false` selects the endpoint as the new insertion point.

TextRange.compareEndPoints() (Method)

Compare two `TextRange` objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.compareEndPoints (*anOperator*, *aTextRange*)
Argument list:	*anOperator*	A selector for the kind of comparison
	aTextRange	A reference to another `TextRange` object

The typical means of comparing two objects is to pass a reference to one object as an argument to a method invoked on the other. This method performs such a comparison. The first keyword is an operator to indicate the kind of comparison to be carried out. The second argument is a reference to the second `TextRange` object that it is to be compared with.

The following keywords can be passed in the first argument as string values:

- `StartToEnd`
- `StartToStart`
- `EndToStart`
- `EndToEnd`

The values returned by this method are -1, 0 or 1. These indicate whether the first value is before, coincident with or after the second value. The operator dictates which values are to be tested.

See also:	`TextRange.isEqual()`,`TextRange.setEndPoint()`

TextRange.duplicate() (Method)

Duplicate a `TextRange` object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`TextRange object`	
JavaScript syntax:	IE	*myTextRange*.duplicate()

This is a deep copy of a `TextRange` object. Simply assigning the value of one variable copies only the reference to the same object. You need to call this method to create a new object whose properties are set to the same values as its parent.

TextRange.execCommand() (Method)

Part of an MSIE special document command handling mechanism. Execute a command across the text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextRange*.execCommand(*aCommand*)
	IE	*myTextRange*.execCommand(*aCommand*, *aUIFlag*)
	IE	*myTextRange*.execCommand(*aCommand*, *aUIFlag*, *aParameter*)
Argument list:	*aCommand*	An MSIE command to execute
	aParameter	Parameter value to the command
	aUIFlag	Display or inhibit UI appearance

The MSIE browser supports a special command handling interface that hooks through the browser's user interface. It allows you to automate user actions in a way that other browsers and non-Windows platform users cannot take advantage of. It is another example of the Microsoft Embrace, Enhance and Eliminate approach to dominating the browser marketplace.

Although this is a method that can be applied to a `document` object, many of the commands that are executed through this mechanism will require that a `TextRange` object is created and available first.

The result returned by this method is a Boolean `true` if the action succeeded and a Boolean `false` if it failed in some way.

The flag parameter provides a way to suppress any user interface changes that may appear as a result of executing the command.

Name	Description
2D-Position	Absolutely positioned elements can be moved by dragging.
AbsolutePosition	Sets an element's position property to "absolute".
BackColor	The background color for the current selection is set to the color value passed in the parameter argument.
Bold	The selected text has `<B>` and `</B>` tags placed at either end.
Copy	The `TextRange` is copied to the clipboard.
CreateBookmark	Carries out modifications to an existing `<A>` tag or creates one, then adds the item to the bookmarks list. The parameter provides the `NAME` value. The `<A>` tag is removed if there is no parameter.
CreateLink	Wraps an `<A HREF="...">` tag around the selected text. The parameter contains the URL value for the HREF.
Cut	Performs a cut to clipboard.
Delete	The text range is deleted. This is not the same as a Cut command.

Table continued on following page

Name	Description
FontName	Wraps <FONT> tags round the selection. The required font face is passed in the parameter.
FontSize	Wraps <FONT> tags round the selection and defines the font's size from the parameter value.
ForeColor	Redefines the foreground text color for the selection taking the color value from the parameter.
FormatBlock	Wraps a <BLOCK> tag round the TextRange.
Indent	The TextRange is indented.
InsertButton	A <BUTTON> tag is placed at the current insertion point in the document. Its ID value is defined by the parameter.
InsertFieldset	A <FIELDSET> tag is inserted with the ID value being taken from the parameter.
InsertHorizontalRule	An <HR> tag is added at the current insertion point.
InsertIFrame	A new <IFRAME> is inserted with the content URL being provided in the parameter.
InsertImage	Overwrites an image on the current selection.
InsertInputButton	An <INPUT TYPE="Button"> is added with its ID value coming from the parameter.
InsertInputCheckbox	An <INPUT TYPE="Checkbox"> tag is added with its ID value coming from the parameter.
InsertInputFileUpload	An <INPUT TYPE="FileUpload"> is added with its ID value coming from the parameter.
InsertInputHidden	An <INPUT TYPE="Hidden"> is added with its ID value coming from the parameter.
InsertInputImage	An <INPUT TYPE="Image"> is added with its ID value coming from the parameter.
InsertInputPassword	An <INPUT TYPE="Password"> is added with its ID value coming from the parameter.
InsertInputRadio	An <INPUT TYPE="Radio"> is added with its ID value coming from the parameter.
InsertInputReset	An <INPUT TYPE="Reset"> is added with its ID value coming from the parameter.
InsertInputSubmit	An <INPUT TYPE="Submit"> is added with its ID value coming from the parameter.
InsertInputText	An <INPUT TYPE="Text"> is added with its ID value coming from the parameter.
InsertMarquee	A new <MARQUEE> is added with the ID being taken from the parameter.
InsertOrderedList	A new <OL> is added with the ID being taken from the parameter.
InsertParagraph	A new <P> is added with the ID being taken from the parameter.
InsertSelectDropdown	A new <SELECT TYPE="select-one"> is added with the ID being taken from the parameter.
InsertSelectListbox	A new <SELECT TYPE="select-multiple"> is added with the ID being taken from the parameter.

Table continued on following page

Name	Description
InsertTextArea	A new <TEXTAREA> is added with the ID being taken from the parameter.
InsertUnorderedList	A new <UL> is added with the ID being taken from the parameter.
Italic	The TextRange is enclosed with <I> tags.
JustifyCenter	The TextRange is centered within its parent object.
JustifyFull	The TextRange is fully justified.
JustifyLeft	The TextRange is left justified.
JustifyRight	The TextRange is right justified.
LiveResize	Causes the MSHTML Editor to update an element's appearance continuously during a resizing or moving operation, rather than updating only at the completion of the move or resize.
MultipleSelection	Allows for the selection of more than one site selectable element at a time when the user holds down the SHIFT or CTRL keys.
Outdent	The complement of the Indent command.
OverWrite	The input-typing mode is set to overwrite if the parameter value is true and insert if it is false.
Paste	The contents of the clipboard are pasted into the TextRange.
PlayImage	If an image represents a video clip, then it starts playing.
Refresh	The document is reloaded.
RemoveFormat	The complement of the FormatBlock command.
SaveAs	Saves the current web page to a file.
SelectAll	The entire document text is selected.
StopImage	The complement of the PlayImage command.
UnBookmark	The complement of the CreateBookmark command.
Underline	Places <U> tags around the TextRange.
Unlink	The complement of the CreateLink command.
Unselect	Unselects whatever was selected to create the TextRange. Many commands are now inappropriate until a new TextRange has been created.

None of these commands provides any greatly significant functionality as far as dynamic HTML is concerned. A few of them allow you to manage the clipboard and bookmark lists. It is probably best to avoid using these commands and use the more usual ways of accessing the document internals.

See also:	Document.execCommand(),FileUpload.select(), TextRange.queryCommandEnabled(), TextRange.queryCommandIndeterm(), TextRange.queryCommandState(), TextRange.queryCommandSupported(), TextRange.queryCommandText(), TextRange.queryCommandValue()

TextRange.expand() (Method)

Expands a `TextRange` by a character, word, sentence or story.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.expand(*aSelector*)
Argument list:	*aSelector*	Indicates what to expand the `TextRange` by

This method has a single argument. The argument indicates what to look for when expanding the `TextRange` object. The end point is modified according to the rules determined by this keyword. The following keywords can be applied:

- `character`
- `word`
- `sentence`
- `textedit`

The `character` keyword causes the endpoint to be indexed onwards by a single character position.

The `word` keyword looks for the next word break in the document text. It also moves the start point to he beginning of the word.

The `sentence` keyword looks for the next full stop at the end of a sentence. It also looks onwards from this to move the start point to the beginning of the sentence.

The `textedit` keyword restores the `TextRange` so it encapsulates only the original selection.

This method returns a Boolean `true` if the range expansion succeeded and a Boolean `false` if it didn't.

See also:	`TextRange.move()`

TextRange.findText() (Method)

Defines a text range according to a search.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`TextRange object`	
JavaScript syntax:	IE	*myTextRange*.findText(*aString*)
Argument list:	*aString*	A string to find in the `TextRange` object

This is a case-insensitive search for a matching text. If the text is located, then the start and end points of the `TextRange` object are adjusted to encapsulate the found text chunk.

The `TextRange` needs to be collapsed for this to work. It could be collapsed and made to point at the start of the document if you want to search the whole text, otherwise the search commences at the current insertion point.

This method returns a Boolean `true` if it found a match and a Boolean `false` if it didn't.

TextRange.getBookmark() (Method)

Bookmarks a position in a text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`TextRange bookmark`	
JavaScript syntax:	IE	*`myTextRange`*`.getBookmmark()`

These are not user bookmarks that are stored in the favorites menu but are special bookmarks for remembering access points in `TextRange` objects.

This method generates a special value that can be stored in a variable and can later be passed to the `moveToBookmark()` method to restore the start and end points of the `TextRange` to the current values. It's a simple way to store and restore the settings of a `TextRange`.

See also:	`TextRange.moveToBookmark()`

TextRange.getBoundingClientRect() (Method)

A method that returns a rectangle measured in pixels within the client display surface.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Rect object`	
JavaScript syntax:	IE	*`myTextRange`*`.getBoundingClientRect()`

This method evaluates the current start and end points of the text range and works out the pixel locations of a bounding rectangle in the display screen. These are then used to instantiate a `textRectangle` object which is returned with the extent rectangle of the `TextRange` object as its values.

A `TextRange` may describe several discontinuous blocks of text. Each one of these would be bounded by a separate client rectangle. This extent rectangle bounds the entire set.

See also:	`Rect` object, `textRectangle` object

TextRange.getClientRects() (Method)

A collection of `textRectangle` objects within the client display surface.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Collection` object	
JavaScript syntax:	IE	*myTextRange*`.getClientRects()`

A `TextRange` may describe several discontinuous blocks of text. Each one of these would be bounded by a separate client rectangle.

This method evaluates the current start and end points of the text range and works out the pixel locations of a bounding rectangle in the display screen for each of the different text areas. These are then used to instantiate a `textRectangle` object which is added to a collection.

The collection of individual client `textRectangle` objects is then returned to the caller.

See also:	`textRectangle` object

TextRange.htmlText (Property)

Returns a text range as HTML source.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextRange*`.htmlText`

This is somewhat similar to the `innerHTML` of an `Element` object. In this case the start and end points of the `TextRange` are taken as the delimiting boundaries. The HTML contained within those boundaries can be returned to your script when the property is read.

Because this property is read-only, you will need to use the `pasteHTML()` method to replace the HTML bounded by a `TextRange` object.

See also:	`TextRange.pasteHTML()`,`TextRange.text`

Property attributes:

`ReadOnly`.

TextRange.inRange() (Method)

Tests for one text range within another.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextRange*`.inRange(`*aTextRange*`)`
Argument list:	*aTextRange*	Another `TextRange` to test against

Given that you have two distinctly separate `TextRange` objects, you can test whether one is contained within the other by passing it as an argument to this method.

TextRange.isEqual() (Method)

Tests two text ranges for equality.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextRange*`.isEqual(aTextRange)`
Argument list:	*aTextRange*	Another `TextRange` to compare against

Given that you have two distinctly separate `TextRange` objects, you can test whether they are both equivalent to one another by passing one as an argument to this method being called on the other. You may accomplish the same thing less conveniently with the `compareEndPoints()` method.

See also:	`TextRange.compareEndPoints()`

TextRange.move() (Method)

Relocates the insertion point of a `TextRange`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*`.move(`*aSelector*`)`
	IE	*myTextRange*`.move(`*aSelector*`, `*aCount*`)`
Argument list:	*aCount*	Indicates how many times to iterate
	aSelector	A rule selector

The `TextRange` is collapsed to an insertion point at the end of the current bounded range. Then the keyword determines how the insertion point is indexed onwards or backwards.

This method has two arguments. The first argument indicates what to look for when moving the `TextRange` object. The second indicates how many times to apply that location search. A negative value indicates a backwards search. The insertion point is modified according to the rules determined by this keyword. If the second argument is omitted, it is assumed to be the value 1. The following keywords can be applied:

- `character`
- `word`
- `sentence`
- `textedit`

The `character` keyword causes the insertion point to be indexed onwards by a single character position.

The `word` keyword looks for the next word break in the document text.

The `sentence` keyword looks for the next full stop at the end of a sentence.

The `textedit` keyword moves the insertion point either to the beginning or end of the original selection.

This method returns an integer describing how many times it was able to move the insertion pointer.

See also:	`TextRange.expand()`

TextRange.moveEnd() (Method)

Relocates the end point of a `TextRange`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.moveEnd(*aSelector*)
	IE	*myTextRange*.moveEnd(*aSelector*, *aCount*)
Argument list:	*aCount*	Indicates how many times to iterate
	aSelector	A rule selector

This works in almost the same way as the `move()` method but instead of collapsing the `TextRange` first, it just operates on the end point of the current text range. Then the keyword determines how the end point is indexed onwards or backwards.

This method has two arguments. The first argument indicates what to look for when adjusting the `TextRange` object. The second indicates how many times to apply that location search. A negative value indicates a backwards search. The end point is modified according to the rules determined by this keyword. If the second argument is omitted, it is assumed to be the value 1. The following keywords can be applied:

- `character`
- `word`
- `sentence`
- `textedit`

The `character` keyword causes the endpoint to be indexed onwards by a single character position.

The `word` keyword looks for the next word break in the document text.

The `sentence` keyword looks for the next full stop at the end of a sentence.

The `textedit` keyword moves the end point either to the beginning or end of the original selection.

This method returns an integer describing how many times it was able to move the end point.

Warnings:

- Be careful that the start and end points do not get crossed over. It is possible that the internal logic of the browser would cope with this and fix things up, but it is a likely area where the browser may be expected to fail.

TextRange.moveStart() (Method)

Relocates the start point of a `TextRange`.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.moveStart(*aSelector*)
	IE	*myTextRange*.moveStart(*aSelector*, *aCount*)
Argument list:	*aCount*	Indicates how many times to iterate
	aSelector	A rule selector

This works in almost the same way as the `move()` method but instead of collapsing the `TextRange` first, it just operates on the start point of the current text range. Then the keyword determines how the start point is indexed onwards or backwards.

This method has two arguments. The first argument indicates what to look for when adjusting the `TextRange` object. The second indicates how many times to apply that location search. A negative value indicates a backwards search. The start point is modified according to the rules determined by this keyword. If the second argument is omitted, it is assumed to be the value 1. The following keywords can be applied:

- `character`
- `word`
- `sentence`
- `textedit`

The `character` keyword causes the startpoint to be indexed onwards by a single character position.

The `word` keyword looks for the next word break in the document text.

The `sentence` keyword looks for the next full stop at the end of a sentence.

The `textedit` keyword moves the start point either to the beginning or end of the original selection.

This method returns an integer describing how many times it was able to move the start point.

Warnings:

- Be careful that the start and end points do not get crossed over. It is possible that the internal logic of the browser would cope with this and fix things up, but it is a likely area where the browser may be expected to fail.

TextRange.moveToBookmark() (Method)

Restores a bookmarked TextRange.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.moveToBookmark(*aBookmark*)
Argument list:	*aBookmark*	A bookmark obtained from a `getBookmark()` method call

This is the complement of the `getBookmark()` method. It takes the result of the `getBookmark()` method that would have been stored in a variable. The `TextRange` start and end points are then reset to the value that was originally defined when the bookmark was recorded.

The Boolean value `true` is returned if the bookmarked locations could be restored and a Boolean `false` value if they couldn't. It really depends on what has happened since the bookmark was captured as to whether it can realistically be restored or not.

See also:	`TextRange.getBookmark()`

TextRange.moveToElementText() (Method)

Expands the text range to encompass an HTML element.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.moveToElementText(*anObject*)
Argument list:	*anObject*	A reference to an object in the document

By passing an HTML `Element` object to this method, you can set the start and end points of the `TextRange` object to neatly bound the text of the HTML `Element` object.

TextRange.moveToPoint() (Method)

Expand the text range to include an x, y location.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.moveToPoint(*anX*, *aY*)
Argument list:	*anX*	An X coordinate value
	aY	A Y coordinate value

The `TextRange` can be set to create an insertion point in the text that corresponds with the X-Y coordinate point of an onscreen location. This is smart stuff indeed because it would be very difficult to compute the text flow and correlate it with screen coordinates.

The result is that the `TextRange` is set to an insertion point.

TextRange.parentElement() (Method)

A reference to an object that is the next outermost item in the document hierarchy.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Element object`	
JavaScript syntax:	IE	*myTextRange*.parentElement()

Having established that a `TextRange` may correspond to some block of text within the document, this returns a reference to an HTML `Element` object that fully encloses the `TextRange`.

For example, if a `TextRange` was set to bound a couple of `Input` elements within a Form, this method might return the `Form` object because that is the next outermost logical object. Of course you could get a `TABLE` object if the form contained a table with the input elements inside it.

TextRange.pasteHTML() (Method)

Pastes HTML or plain text into the text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*.pasteHTML(*someHTML*)
Argument list:	*someHTML*	A string containing valid HTML

Because the `htmlText` property is a read-only value, you will need to use this method to replace the HTML bounded by the `TextRange` or to insert HTML if the `TextRange` describes an insertion point.

See also:	`TextRange.htmlText`

TextRange.queryCommandEnabled() (Method)

Part of an MSIE special document command handling mechanism. Indicates the disposition of the specified command.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextRange*.queryCommandEnabled (*aCommandName*)
Argument list:	*aCommandName*	An MSIE command name

This method returns a Boolean value that indicates whether the named command is enabled. Many factors can affect the result of this command. It may depend on the ready state of the document or whether a selection is in force.

Refer to the `TextRange.execCommand()` method for a list of the available commands.

See also:	`Document` object, `Document.queryCommandEnabled()`, `TextRange.execCommand()`

TextRange.queryCommandIndeterm() (Method)

Part of an MSIE special document command handling mechanism. Indicates whether the command is in an indeterminate state.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextRange*.queryCommandIndeterm (*aCommandName*)
Argument list:	*aCommandName*	An MSIE command name

If the document is not fully loaded (you can check the `readyState`), or if a command might not be available due to some of its prerequisites not being set (such a selection creating a `TextRange`), this method will return a Boolean `true` value. If it returns a Boolean `false`, then the command may be available as determined by the enabled test.

See also:	`Document` object, `Document.queryCommandIndeterm()`, `TextRange.execCommand()`

TextRange.queryCommandState() (Method)

Part of an MSIE special document command handling mechanism. Returns the current state of a command.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive` or `Null`	
JavaScript syntax:	IE	*myTextRange*`.queryCommandState(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

This will return one of the following values:

- Boolean `true` if the command has completed.
- Boolean `false` if it is still in progress.
- `Null` if the state cannot be determined.

See also:	`Document` object, `Document.queryCommandState()`, `TextRange.execCommand()`

TextRange.queryCommandSupported() (Method)

Part of an MSIE special document command handling mechanism. Checks to see if a command is supported.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextRange*`.queryCommandSupported(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

Some commands are not supported by the `TextRange` object but may be supported by the `document` object.

This method returns a Boolean `true` value if the command is supported by the `TextRange` object.

See also:	`Document` object, `Document.queryCommandSupported()`, `TextRange.execCommand()`

TextRange.queryCommandText() (Method)

Part of an MSIE special document command handling mechanism. Returns the string associated with a command.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextRange*`.queryCommandText(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

Some commands support the extraction of text from the `document` or `TextRange`. If the command does support the extraction of text, it will be returned by this method.

See also:	`Document` object, `Document.queryCommandText()`, `TextRange.execCommand()`

TextRange.queryCommandValue() (Method)

Part of an MSIE special document command handling mechanism. Returns the value of the command.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextRange*`.queryCommandValue(`*aCommandName*`)`
Argument list:	*aCommandName*	An MSIE command name

The value of a command depends on the command itself and what is selected. This method returns a value according to those criteria.

See also:	`Document` object, `Document.queryCommandValue()`, `TextRange.execCommand()`

TextRange.select() (Method)

Select the text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*`.select()`

If the browser supports a `Selection` object or `TextRange` objects, you may then be able to access the selected text using JavaScript. Of course in a form object, the text of the whole object can also be accessed, but this may not be what was selected because the user may select all or part of a page, and that selection may span several form elements or only part of a form element.

Unless you call this method, the bounded area of the `TextRange` object will not be visible to the user.

TextRange.setEndPoint() (Method)

Set the end point of a text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextRange*`.setEndPoint(`*anOperator*`, `*aTextRange*`)`
Argument list:	*anOperator*	A selector for the kind of copy
	aTextRange	A reference to another `TextRange` object

This method provides a way to copy start and end points between distinctly different `TextRange` objects. The first keyword is an operator to indicate the kind of copy to be carried out. The second argument is a reference to the second `TextRange` object that is to act as a source for the new values. Although this method is called `setEndPoint()`, it will allow both start and end points to be modified.

The following keywords can be passed in the first argument as string values:

- `StartToEnd`
- `StartToStart`
- `EndToStart`
- `EndToEnd`

There are no values returned by this method. The indicated end points are simply copied from the source range to the target range.

See also:	`TextRange.compareEndPoints()`

TextRange.text (Property)

Extract the text of a text range.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextRange*`.text`

This is somewhat similar to the `innerText` of an `Element` object. In this case the start and end points of the `TextRange` are taken as the delimiting boundaries. The text contained within those boundaries can be returned to your script when the property is read. If there are any HTML tags within the bounded extent of the `TextRange`, they will be filtered out so that the method yields pure text only values.

Because this property is read-only, you will need to use the `pasteHTML()` method to replace the HTML bounded by a `TextRange` object. However, because there appears not to be a `pasteText()` method, you may need to do some complex reconstructive work if there was embedded HTML and you only requested the text.

See also:	`TextRange.htmlText`

textRectangle object (Object/JScript)

The extent rectangle that encloses a `TextRange` object.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
JavaScript syntax:	IE	*myTextRectangle* = *myTextRange*.getBoundingClientRect()
Object properties:	bottom, left, right, top	

This is a close relation to the `rect` object. This is a special case, used for describing rectangles on the screen which are the bounding extent rectangles for `TextRange` objects.

You shouldn't try to modify the properties of this object directly. It's intended for you to read to establish where on the screen the `TextRange` is located.

See also:	`Clip` object, `Rect` object, `TextRange.getBoundingClientRect()`, `TextRange.getClientRects()`

Property	JavaScript	JScript	N	IE	Opera	Notes
bottom	-	5.0 +	-	5.0 +	-	-
left	-	5.0 +	-	5.0 +	-	-
right	-	5.0 +	-	5.0 +	-	-
top	-	5.0 +	-	5.0 +	-	-

textRectangle.bottom (Property)

The bottom of a `textRectangle`.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	Number primitive	
JavaScript syntax:	IE	*myTextRectangle*.bottom

See also:	`Rect.bottom`, `textRectangle` object

textRectangle.left (Property)

The left edge of a `textRectangle`.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myTextRectangle`*`.left`

See also:	`Rect.left`, `textRectangle` object

textRectangle.right (Property)

The right edge of a text rectangle.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myTextRectangle`*`.right`

See also:	`Rect.right`, `textRectangle` object

textRectangle.top (Property)

The top edge of a `textRectangle`.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myTextRectangle`*`.top`

See also:	`Rect.top`, `textRectangle` object

TextStream object (Object/JScript)

An object that represent an I/O text stream. Very useful in a server-side context.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextStream* = *myFile*.OpenAsTextStream()
	IE	*myTextStream* = *myFileSystem*.OpenTextFile (*aName*, *aMode*, *aFlag*, *aFormat*)
Argument list:	*aName*	The name of the file to be created
	aFlag	A flag indicating whether the file can be created if necessary
	aMode	An access mode for the file
	aFormat	A format control for the file
Object properties:	AtEndOfLine, AtEndOfStream, Column, Line	
Object methods:	Close(), Read(), ReadAll(), ReadLine(), Skip(), SkipLine(), Write(), WriteBlankLines(), WriteLine()	

This object is a wrapper for a file when opened for I/O. With this object you can read and write to the file.

Files can be opened via methods belonging to the `File` object or by requesting that the `FileSystem` object open a named file. Both techniques yield the same kind of `TextStream` object.

See also:	Active Server Pages, File.OpenAsTextStream(), FileSystem.OpenTextFile()

Property	JavaScript	JScript	N	IE	Opera	Notes
AtEndOfLine	-	2.0 +	-	4.0 +	-	-
AtEndOfStream	-	2.0 +	-	4.0 +	-	-
Column	-	2.0 +	-	4.0 +	-	-
Line	-	2.0 +	-	4.0 +	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
Close()	-	2.0 +	-	4.0 +	-	-
Read()	-	2.0 +	-	4.0 +	-	-
ReadAll()	-	2.0 +	-	4.0 +	-	-
ReadLine()	-	2.0 +	-	4.0 +	-	-
Skip()	-	2.0 +	-	4.0 +	-	-
SkipLine()	-	2.0 +	-	4.0 +	-	-
Write()	-	2.0 +	-	4.0 +	-	-
WriteBlankLines()	-	2.0 +	-	4.0 +	-	-
WriteLine()	-	2.0 +	-	4.0 +	-	-

TextStream.AtEndOfLine (Property)

A Boolean value that indicates the status of the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextStream*`.AtEndOfLIne`

This property returns a value `true` if the file pointer is immediately preceding an end of line character.

TextStream.AtEndOfStream (Property)

A Boolean value that indicates the status of the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	*myTextStream*`.AtEndOfStream`

This property returns the Boolean value `true` if the file pointer is at the end of the text stream.

TextStream.Close() (Method)

A method that closes a text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextStream*`.Close()`

The text stream is closed and any subsequent access will generate an error. You will need to re-open the text stream to gain access to it again.

TextStream.Column (Property)

The column number within the file.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myTextStream*`.Column`

This property facilitates the use of fixed length record structured files via a stream.

It yields the character column number within the current line.

TextStream.Line (Property)

The line number within the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myTextStream*.Line

The line number property is incremented for every newline character that is encountered within the file. This is effectively the record number within the file.

TextStream.Read() (Method)

A method that reads text from the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextStream*.Read(*aCount*)
Argument list:	*aCount*	A count of characters to read

This will read the specified number of characters from the file. Note that this may span a line break and is intended for reading fixed length records.

TextStream.ReadAll() (Method)

A method that reads the entire text stream in one go.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextStream*.ReadAll()

The entire text stream is returned in a single read. Be careful if you are dealing with extraordinarily large files. You may get back more data than you expected.

TextStream.ReadLine() (Method)

A method that reads a line from the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTextStream*`.ReadLine()`

This method will read the remainder of a line up to but not including the newline character.

TextStream.Skip() (Method)

A method that skips a specified number of characters through the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextStream*`.Skip(`*aCount*`)`
Argument list:	*aCount*	The number of characters to skip through the file

Refer to:

`TextStream.Read()`

TextStream.SkipLine() (Method)

A method that skips a line of the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextStream*`.SkipLine()`

Refer to:

`TextStream.ReadLine()`

TextStream.Write() (Method)

A method that writes to the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextStream*.Write(*aString*)
Argument list:	*aString*	Some text to write to the file

This writes exactly (and only) the text passed in its string argument.

See also:	TextStream.WriteLine()

TextStream.WriteBlankLines() (Method)

A method that writes a specified number of blank lines to the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextStream*.WriteBlankLines(*aCount*)
Argument list:	*aCount*	The number of blank lines to write to the file

This writes just the newline characters. The count value specifies just how many.

See also:	TextStream.WriteLine()

TextStream.WriteLine() (Method)

A method that writes a line to the text stream.

Availability:	JScript – 2.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	*myTextStream*.WriteLine()
	IE	*myTextStream*.WriteLine(*aString*)
Argument list:	*aString*	A string to be written to the file

This is similar to the Write() method except that it automatically places a newline character after the string that has been written. In addition, the string value is optional in which case this is functionally identical to calling WriteBlankLines(1).

See also:	TextStream.Write(),TextStream.WriteBlankLines()

TFOOT object (Object/HTML)

An object that encapsulates a <TFOOT> tag within a <TABLE> block.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myTFOOT* = *myDocument*.all.*anElementID*
	IE	*myTFOOT* = *myDocument*.all.tags("TFOOT")[*anIndex*]
	IE	*myTFOOT* = *myDocument*.all[*aName*]
	-	*myTFOOT* = *myDocument*.getElementById(*anElementID*)
	-	*myTFOOT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myTFOOT* = *myDocument*.getElementsByTagName ("TFOOT")[*anIndex*]
HTML syntax:	<TFOOT> ... </TFOOT>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align`, `bgColor`, `ch`, `chOff`, `vAlign`	
Object methods:	`deleteRow()`, `insertRow()`	
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	
Collections:	`rows[]`	

A table must contain one and only one `TFOOT` object. If you don't create one automatically, the `TABLE` object instantiates one for you but it would be empty.

The `TFOOT` object is instantiated by a <TFOOT> HTML tag. This is a means of marking off a section at the bottom of the table so that the rows can be grouped together and operated on separately to the table body.

The DOM level 1 standard calls for the implementation of a `TableSectionElement` object which includes both `TFOOT` and `THEAD` in its capabilities.

Warnings:

- Some earlier versions of MSIE for Macintosh have very limited capabilities implemented for this object. You cannot access any of the HTML or text contained in the object, nor the `rows` collection.

See also:	`Element` object, `TABLE` object, `TABLE.createTFoot()`, `TABLE.deleteTFoot()`, `TABLE.rules`, `TABLE.tFoot`, `THEAD` object, `TR` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`bgColor`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`ch`	1.5 +	-	6.0 +	-	-	1 +	-	-
`chOff`	1.5 +	-	6.0 +	-	-	1 +	-	-
`vAlign`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`deleteRow()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`insertRow()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

TFOOT.align (Property)

The kind of horizontal alignment applied to items within the `<TFOOT>` block of a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTFOOT*`.align`

The horizontal alignment of the `TFOOT` object with respect to its containing parent object is defined in this property. The available set of alignment specifiers are:

- `center`
- `left`
- `right`
- `char`
- `justify`

See also:	`TABLE.align`, `TBODY.align`, `TD.align`, `TH.align`, `THEAD.align`, `TR.align`

TFOOT.bgColor (Property)

The background color for items in the `<TFOOT>` block.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTFOOT*`.bgColor`

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	Color names, Color value, `THEAD.bgColor`

TFOOT.ch (Property)

The character used for alignment of columns within the table.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTFOOT*.ch

TFOOT.chOff (Property)

The offset of character alignments within a column.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTFOOT*.chOff

TFOOT.deleteRow() (Method)

In a multiple row footer, you can delete a particular row with this method.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTFOOT*.deleteRow(*anIndex*)
Argument list:	*anIndex*	The row to delete

See also:	THEAD.deleteRow()

TFOOT.insertRow() (Method)

You can insert additional rows into a table footer with this method.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`TR object`	
JavaScript syntax:	-	*myTFOOT*`.insertRow(`*anIndex*`)`
Argument list:	*anIndex*	The row at which to place the new inserted `TR` object

See also:	`THEAD.insertRow()`

TFOOT.rows[] (Collection)

A collection of rows within a `<TFOOT>` block belonging to a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	-	*myTFOOT*`.rows`

This is not the same as the `rows` collection returned from a `TABLE` object. That is because the `rows[]` collection belonging to a `TFOOT` object can only list those `TR` objects that are contained within the `<TFOOT>` tags.

See also:	Collection object, `TABLE.cells[]`, `TABLE.cols`, `TABLE.rows[]`, `TBODY.rows[]`, `THEAD.rows[]`, `TR.rowIndex`

Property attributes:

`ReadOnly`.

TFOOT.vAlign (Property)

The vertical alignment applied to items within the <TFOOT> block of a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTFOOT*.vAlign

The vertical alignment of the content within the table cells can be controlled across an entire TFOOT extent with this property. The following keywords can be assigned to it:

- baseline
- bottom
- middle
- top

These may be also available on some implementations:

- absbottom
- absmiddle
- baseline
- texttop

See also:	TBODY.vAlign, TD.vAlign, TH.vAlign, THEAD.vAlign, TR.vAlign

TH object (Object/HTML)

An object that encapsulates a <TH> table header cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Inherits from:	Element object

JavaScript syntax:	IE	*myTH* = *myDocument*.all.*anElementID*
	IE	*myTH* = *myDocument*.all.tags("TH")[*anIndex*]
	IE	*myTH* = *myDocument*.all[*aName*]
	-	*myTH* = *myDocument*.getElementById(*anElementID*)
	-	*myTH* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myTH* = *myDocument*.getElementsByTagName("TH")[*anIndex*]
HTML syntax:	<TH> ... </TH>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
Object properties:	abbr, align, axis, background, bgColor, borderColor, borderColorDark, borderColorLight, cellIndex, ch, chOff, colSpan, headers, height, noWrap, rowSpan, scope, vAlign, width	
Event handlers:	onAfterUpdate, onBeforeUnload, onBlur, onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onResize, onRowEnter, onRowExit, onSelectStart	

This object is instantiated by a <TH> tag that encloses the content of a single data cell. This cell is formatted differently to a TD cell because a TH cell is considered to be a table header cell whereas a TD cell is a table data cell.

Some of the property values in this object may be inherited from parent objects such as TABLE, TR and TBODY.

See also:	Element object, TABLE object, TD object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
abbr	1.5 +	-	6.0 +	-	-	1 +	-	-
align	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
axis	1.5 +	-	6.0 +	-	-	1 +	-	-
background	-	3.0 +	-	4.0 +	-	-	-	-
bgColor	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
borderColor	-	3.0 +	-	4.0 +	-	-	-	-
borderColorDark	-	3.0 +	-	4.0 +	-	-	-	-
borderColorLight	-	3.0 +	-	4.0 +	-	-	-	-
cellIndex	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
ch	1.5 +	-	6.0 +	-	-	1 +	-	-
chOff	1.5 +	-	6.0 +	-	-	1 +	-	-
colSpan	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
headers	1.5 +	-	6.0 +	-	-	1 +	-	-
height	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
noWrap	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	Warning
rowSpan	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
scope	1.5 +	-	6.0 +	-	-	1 +	-	-
vAlign	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
width	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterUpdate	-	3.0 +	-	4.0 +	-	-	-	-
onBeforeUnload	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
onClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onResize	1.5 +	3.0 +	6.0 +	4.0 +	-	-	-	Warning
onRowEnter	-	3.0 +	-	4.0 +	-	-	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-	-	-
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

TH.abbr (Property)

An abbreviation value to be used for header cells in the column where the data cell resides.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTH*.abbr

See also:	`TD.abbr`

TH.align (Property)

The alignment of content within a `<TH>` table cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTH*.align

The alignment of the `TH` object with respect to its containing parent object is defined in this property. The available set of alignment specifiers are:

- `center`
- `left`
- `right`
- `char`
- `justify`

See also:	`TABLE.align`, `TBODY.align`, `TD.align`, `TFOOT.align`, `THEAD.align`, `TR.align`

TH.axis (Property)

The names group of related header cells.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTH*`.axis`

See also:	`TD.axis`

TH.background (Property)

A URL for an image to load as the background for a table header cell.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTH*`.background`

If a background image is available, then its URL is contained in this property. Changing the value in this property will replace the background with a new one. However there may be a perceptible delay while the new image is fetched from the web server.

See also:	`TD.background`

TH.bgColor (Property)

The background color for a table header cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTH*`.bgColor`

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	Color names, Color value, `TD.bgColor`

TH.borderColor (Property)

The border color around a table header cell.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTH*`.borderColor`

You can use this property to determine the color of a border surrounding a table header cell. Note that there are additional properties to determine the highlights and lowlights of the table header cell border coloring.

See also:	`TD.borderColor`

TH.borderColorDark (Property)

The color value of the shadowed edge of the table cell border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTH*`.borderColorDark`

Table header cell borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the shadowed part of the table header cell border.

See also:	`TD.borderColorDark`

TH.borderColorLight (Property)

The color value of the highlighted edge of the table cell border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTH*`.borderColorLight`

Table header cell borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the highlighted part of the table header cell border.

See also:	`TD.borderColorLight`

TH.cellIndex (Property)

A zero-based integer number that indicates the position of a `<TH>` cell within a `<TR>` row. This is effectively the horizontal column coordinate of a table header cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTH*`.cellIndex`

You can access cells by means of the `rows[]` and `cells[]` collections belonging to the `TABLE` and `TR` objects respectively. This property provides the horizontal coordinate to use in the `TR.cells[]` collection to access the object describing this table cell.

See also:	`TD.cellIndex`

Property attributes:

`ReadOnly`.

TH.ch (Property)

The alignment character for cells in a column arrangement.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTH*.ch

HTML 4.0 provides for text to be arranged in neat columns within table cells without the need to create additional tables within tables. This method of alignment is selected by setting the `ALIGN="CHAR"` HTML tag attribute. The `CHAR` HTML tag attribute is reflected in this property and is active when the `CHAROFF` HTML tag attribute is present.

See also:	`COL.ch`, `COLGROUP.ch`, `TD.ch`, `THEAD.ch`, `TR.ch`

TH.chOff (Property)

The offset of a column alignment character.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTH*.chOff

The `CHAR` alignment style requires that an alignment character be specified with the `ch` property and that an offset measured in pixels be defined as its value. The offset value can be defined with the `CHAROFF` HTML tag attribute. The remainder of the string is offset by this distance from the alignment character.

See also:	`COL.chOff`, `COLGROUP.chOff`, `TD.chOff`, `THEAD.chOff`, `TR.chOff`

TH.colSpan (Property)

The number of columns that a header cell spans.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTH*`.colSpan`

This corresponds to the `COLSPAN` attribute within a `<TH>` HTML tag for a table header cell description. It defines how many table columns this column is to span.

Warnings:

- Note that this may affect the value of the `cellIndex` property for subsequent cells within the same row.

See also:	`COL.span`, `COLGROUP.span`, `style.columnSpan`, `TD.colSpan`

TH.headers (Property)

A list of `ID` attribute values for header cells.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTH*`.headers`

See also:	`TD.headers`

TH.height (Property)

The height in pixels of the table header cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`String primitive`
JavaScript syntax:	- *myTH*`.height`

The table header cell space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the height of that extent rectangle.

See also:	`TD.height`

TH.noWrap (Property)

Controls whether text in the table header cell is allowed to wrap.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0
Property/method value type:	`Boolean primitive`
JavaScript syntax:	- *myTH*`.noWrap`

This is a Boolean value that controls whether the textual content is wrapped at the right hand window border or not.

If the value `false` is assigned to this property, then words will wrap as the page is drawn. This is good and is the way you would expect a browser to behave. The text will flow according to the space available.

If the value `true` is assigned to this property, the line of text will continue to the right until a `<BR>` or other block level tag is encountered. This will force the horizontal width of the page to be extremely large and the user will need to scroll furiously to be able to see the text and then scroll back again for the start of the next line.

Warnings:

- Only use this if you plan to place line breaks at frequent intervals yourself and really do need to control the line breaks manually.

See also:	`TD.noWrap`

TH.rowSpan (Property)

The number of rows that the table header will span.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTH*`.rowSpan`

This is used when you want to create complex tables. This technique may be an alternative to nesting a table. That is always good because sometimes nested tables can become very unwieldy.

See also:	`style.rowSpan`, `TD.rowSpan`

TH.scope (Property)

The scope covered by header cells.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTH*`.scope`

See also:	`TD.scope`

TH.vAlign (Property)

The vertical alignment of content within the table header cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTH*.vAlign

The vertical alignment of the content within the table cells can be controlled for a single table cell with this property. The following keywords can be assigned to it:

- `baseline`
- `bottom`
- `middle`
- `top`

These may be also available on some implementations:

- `absbottom`
- `absmiddle`
- `baseline`
- `texttop`

This value will override the setting for a row or `TBODY` extent.

See also:	`TBODY.vAlign`, `TD.vAlign`, `TFOOT.vAlign`, `THEAD.vAlign`, `TR.vAlign`

TH.width (Property)

The width in pixels of the table header cell.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTH*.width

The table header cell space is defined by an extent rectangle that surrounds the space occupied by it on the screen. An extent rectangle is that smallest rectangle that completely encloses the item. This property specifies the width of that extent rectangle.

See also:	`TD.width`

THEAD object (Object/HTML)

An object that encapsulates a `<THEAD>` tag within a `<TABLE>` block.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myTHEAD* = *myDocument*.all.*anElementID*
	IE	*myTHEAD* = *myDocument*.all.tags("THEAD")[*anIndex*]
	IE	*myTHEAD* = *myDocument*.all[*aName*]
	-	*myTHEAD* = *myDocument*.getElementById(*anElementID*)
	-	*myTHEAD* = *myDocument*.getElementsByName (*aName*)[*anIndex*]
	-	*myTHEAD* = *myDocument*.getElementsByTagName ("THEAD")[*anIndex*]
HTML syntax:	`<THEAD> ... </THEAD>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align`, `bgColor`, `ch`, `chOff`, `vAlign`	
Object methods:	`deleteRow()`, `insertRow()`	
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	
Collections:	`rows[]`	

A table must contain one and only one `THEAD` object. If you don't create one automatically, the `TABLE` object instantiates one for you, but it will be empty.

The `THEAD` object is instantiated by a `<THEAD>` HTML tag. This is a means of marking off a section at the top of the table so that the rows can be grouped together and operated on separately to the table body.

The DOM level 1 standard calls for the implementation of a `TableSectionElement` object which includes both `TFOOT` and `THEAD` in its capabilities.

See also:	`Element` object, `TABLE` object, `TABLE.createTHead()`, `TABLE.deleteTHead()`, `TABLE.rules`, `TABLE.tHead`, `TFOOT` object, `TR` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`bgColor`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`ch`	1.5 +	-	6.0 +	-	-	1 +	-	-
`chOff`	1.5 +	-	6.0 +	-	-	1 +	-	-
`vAlign`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`deleteRow()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`insertRow()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

THEAD.align (Property)

The alignment of items within the <THEAD> block of a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTHEAD*.align

The alignment of the THEAD object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- center
- left
- right
- char
- justify

See also:	TABLE.align, TBODY.align, TD.align, TFOOT.align, TH.align, TR.align

THEAD.bgColor (Property)

The background color of items in the <THEAD> block of a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	String primitive	
JavaScript syntax:	-	*myTHEAD*.bgColor

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	Color names, Color value, TFOOT.bgColor

THEAD.ch (Property)

The alignment character for cells in a column arrangement.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTHEAD*.ch

HTML 4.0 provides for text to be arranged in neat columns within table cells without the need to create additional tables within tables. This method of alignment is selected by setting the `ALIGN="CHAR"` HTML tag attribute. The `CHAR` HTML tag attribute is reflected in this property and is active when the `CHAROFF` HTML tag attribute is present.

See also:	`COL.ch`, `COLGROUP.ch`, `TD.ch`, `TH.ch`, `TR.ch`

THEAD.chOff (Property)

The offset of a column alignment character.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTHEAD*.chOff

The CHAR alignment style requires that an alignment character be specified with the `ch` property and that an offset measured in pixels be defined as its value. The offset value can be defined with the `CHAROFF` HTML tag attribute. The remainder of the string is offset by this distance from the alignment character.

See also:	`COL.chOff`, `COLGROUP.chOff`, `TD.chOff`, `TH.chOff`, `TR.chOff`

THEAD.deleteRow() (Method)

In a multiple row footer, you can delete a particular row with this method.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTHEAD*.deleteRow(*anIndex*)
Argument list:	*anIndex*	The row to be deleted

See also:	TFOOT.deleteRow(),TR.deleteCell()

THEAD.insertRow() (Method)

You can insert additional rows into a table footer with this method.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTHEAD*.insertRow(*anIndex*)
Argument list:	*anIndex*	The row at which to insert the new TR object

See also:	TFOOT.insertRow()

THEAD.rows[] (Collection)

A collection of rows within a <THEAD> block of a table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	Collection object	
JavaScript syntax:	-	*myTHEAD*.rows

This is not the same as the `rows` collection returned from a `TABLE` object. That is because the `rows[]` collection belonging to a `THEAD` object can only list those `TR` objects that are contained within the `<THEAD>` tags.

See also:	Collection object, `TABLE.cells[]`, `TABLE.cols`, `TABLE.rows[]`, `TBODY.rows[]`, `TFOOT.rows[]`, `TR.rowIndex`

Property attributes:

`ReadOnly`.

THEAD.vAlign (Property)

A control for the vertical alignment of cells within the `THEAD` object.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTHEAD*`.vAlign`

The vertical alignment of the content within the table cells can be controlled across an entire `THEAD` extent with this property. The following keywords can be assigned to it:

- `baseline`
- `bottom`
- `middle`
- `top`

These may be also available on some implementations:

- `absbottom`
- `absmiddle`
- `baseline`
- `texttop`

See also:	`TBODY.vAlign`, `TD.vAlign`, `TFOOT.vAlign`, `TH.vAlign`, `TR.vAlign`

this (Keyword)

A reference to the receiving object.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	An object	
JavaScript syntax:	-	this

Every active execution context owns a this value. It is used for self-referring script statements.

The specific this value of an execution context depends on the caller and the type of code being executed. This is determined on entry to an execution context. The this value associated with an execution context is immutable and therefore cannot be changed from a script.

A this value is considered to be a primary expression.

The this keyword is often used inside function bodies that are registered with a prototype. When it is executed, the function can refer to its owning object without having to know what sort of object it is. We can use this to write a function that can be used with several kinds of object.

If the this keyword is used inside an event handler, it refers to the object that the event belongs to. We can exploit this to build event handlers that support many objects and can be called by different event types.

If the this keyword is used outside of all functions (in global code) it refers to the Global object. A this property used in a script will therefore return that global object. In fact it will return an object of type Window.

Other object types will be returned according the context in which the property is applied.

The example shows how the this keyword can be used to enhance the prototype of an object.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// See what the scope rules define for 'this'
document.write(this);
document.write("<BR>");
```

```
document.write(typeof this);
document.write("<BR>");

// Create a user defined prototype
function AnimalClass()
{
   return "Animal";
}

function Animal(aSpecies, aHabitat)
{
   this.species  = aSpecies;
   this.habitat  = aHabitat;
   this.toString = AnimalClass;
}

// Instantiate an animal
myAnimal = new Animal("Cow", "Field");
document.write("Object: "  + myAnimal + "<BR>");
document.write("Species: " + myAnimal.species + "<BR>");
document.write("Habitat: " + myAnimal.habitat + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	Execution context, Method, Primary expression, Reference

Cross-references:

ECMA 262 edition 2 – section – 10.1.6

ECMA 262 edition 2 – section – 10.1.7

ECMA 262 edition 2 – section – 11.1.1

ECMA 262 edition 3 – section – 10.1.6

ECMA 262 edition 3 – section – 10.1.7

ECMA 262 edition 3 – section – 11.1.1

Wrox *Instant JavaScript* – page – 30

Wrox *Instant JavaScript* – page – 53

Thousands separator (Definition)

A special locale specific character for indicating groups of digits.

See also:	Decimal point (.), Localization

throw (Statement)

Throw a custom exception in the hope it will be caught by an error handler.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

The `throw` statement provides a way to create an exception which will be passed to an associated `catch()` handler in a `try ... catch` structure. It is really intended to be used in that context, but you can use it outside of a `try ... catch` structure and trap the exception with the normal `onError` event handling support.

You can use this mechanism to generate an error, perhaps as a result of testing some value. Placing a `throw` statement into your code will force the error handling to be invoked. However if you use it outside of a `try ... catch` block, the browser error handling will generate an error due to there being no way of catching the thrown event. It forces an error dialog, but not the one you wanted. You need to use a `throw` with a `try ... catch` block to force the `catch` code to be called.

You can work around this by assigning an error handler function to the `onerror` property and making sure that the error handler returns a Boolean `true` value to signify that the error has no further processing required.

Warnings:

- This is not supported by Netscape Enterprise Server 3, and so its error handling capabilities are not available server -side.
- When using MSIE version 5 on the Macintosh platform, if you set the `LANGUAGE` HTML tag attribute of the enclosing `<SCRIPT>` tag to JScript 1.3, this handler will not be invoked properly.
- The example below, while supported on Netscape 6.0, does not seem to be supported on Internet Explorer 5.0 or 5.5.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
// Define an error handler function
function myErrHandler(anException)
{
   alert("An error happened and was caught by this handler.");
   return true;
}

// Register the error handler
onerror = myErrHandler;
```

```
// Throw an exception
throw "ERR";
</SCRIPT>
</BODY>
</HTML>
```

See also:	`catch( ... )`, `Error` object, `EvalError` object, Exception handling, `finally ...`, `RangeError` object, `ReferenceError` object, `SyntaxError` object, `try ... catch ... finally`, `TypeError` object, `URIError` object

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

ECMA 262 edition 3 – section – 12.13

throws (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Time from year (Time calculation)

A date and time algorithm defined by ECMAScript.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

To calculate any time values relative to the start of a year, we need to know what instant the year began.

All non-leap years have 365 days with the usual number of days in each month. Leap years have an extra day in February. The calculation shown below uses known leap years and non leap years to adjust the day numbers and yield the day number of the first day of the given year and then use that to work out the time in milliseconds when the year started:

```
DayFromYear(y) = 365 * (y - 1970) + floor((y - 1969) / 4) - floor((y - 1901) /
100) + floor((y - 1601) / 400)

msPerDay = 86400000TimeFromYear(y) = msPerDay * DayFromYear(y)
```

Subtracting this value from any absolute time value gives you a millisecond accurate offset from the beginning of the specified year.

Example code:

```
// Work out days and milliseconds at start of a year
var msPerDay = 86400000;

// Test day from year
document.write("<TABLE BORDER=1>");
for(var ii=1980; ii<2009; ii++)
{
   document.write("<TR>");
   document.write("<TD>");
   document.write(ii);
   document.write("</TD>");
   document.write("<TD>");
   document.write(dayFromYear(ii));
   document.write("</TD>");
   document.write("<TD>");
   document.write(timeFromYear(ii));
   document.write("</TD>");
   document.write("</TR>");
}
document.write("</TABLE>");

// Work out milliseconds at start of year
function timeFromYear(aYear)
{
   var myTime = msPerDay * dayFromYear(aYear);
   return myTime;
}

// Day from year function
function dayFromYear(aYear)
{
   var myDay = 365 * (aYear - 1970)  +
      Math.floor((aYear - 1969) / 4) -
      Math.floor((aYear - 1901) / 100)  +
      Math.floor((aYear - 1601) / 400);
   return myDay;
}
```

See also:	Broken down time, Date from time, Date number, Day from year, Day number, Day within year, Time range, Time within day, Year from time, Year number

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.3

ECMA 262 edition 3 – section – 15.9.1.3

Time range (Definition)

ECMAScript measures time in milliseconds since 01-01-1970 UTC.

Availability:	ECMAScript edition – 2

In ECMA compliant implementations, time is measured in milliseconds since the first of January 1970 UTC.

In ECMA compliant implementations, leap seconds are ignored and it is assumed that there are exactly 86,400,000 milliseconds per day. The available range of number values is 18 quadrillion, which is sufficient to measure, to millisecond accuracy, over a time period of nearly 286,000 years forwards or backwards from 01-January-1970 UTC.

`Date` objects don't use this entire range of values and only cope with 100 million days either side of 01-January-1970 UTC. Still, that is a time period that covers just over half a million years. So, no Y2K crisis there (probably).

The exact moment of midnight at the beginning of 01-January-1970 UTC is represented by the value 0.

The time range may not be the same as that provided by the underlying host environment. For example, Macintosh dates and times are based on a start time of the first of January 1904 measured in seconds. The adjustment is trivial in computational terms but may be missing in some implementations.

See also:	Broken down time, Calendar time, Date from time, Date number, `Date` object, Day from year, Day number, Days in year, In leap year, `MakeDate()`, `MakeDay()`, `MakeTime()`, Month number, Time from year, Time value, Time within day, Week day

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.1

ECMA 262 edition 3 – section – 15.9.1.1

Time value (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

The ECMA standard defines some useful methods for decomposing time values.

Some constant values need to be defined first:

- HoursPerDay = 24
- MinutesPerHour = 60
- SecondsPerMinute = 60
- msPerSecond = 1000

From these we can derive some other values:

- msPerMinute = msPerSecond * SecondsPerMinute = 60000
- msPerHour = msPerMinute * MinutesPerHour = 3600000

Now the methods can be defined in terms of these constants and their derivatives.

- HourFromTime(t) = floor(t / msPerHour) modulo HoursPerDay
- MinFromTime(t) = floor(t / msPerMinute) modulo MinutesPerHour
- SecFromTime(t) = floor(t / msPerSecond) modulo SecondsPerMinute
- msFromTime(t) = t modulo msPerSecond

Example code:

```
// Define some constants
HoursPerDay      =   24;
MinutesPerHour   =   60;
SecondsPerMinute =   60;
msPerSecond      = 1000;

// Derived values
msPerMinute = msPerSecond * SecondsPerMinute;
msPerHour   = msPerMinute * MinutesPerHour;

// Grab the time now in milliseconds
myMilliseconds = Number(new Date());
document.write("Hours ...: ");
document.write(HourFromTime(myMilliseconds));
document.write("<BR>");
document.write("Minutes ...: ");
document.write(MinFromTime(myMilliseconds));
document.write("<BR>");
document.write("Seconds ...: ");
document.write(SecFromTime(myMilliseconds));
document.write("<BR>");
document.write("Milliseconds ...: ");
document.write(msFromTime(myMilliseconds));
document.write("<BR>");

// Work out the hour number (Add 1 hour for BST)
function HourFromTime(aMillisecondTime)
{
```

```
   return (Math.floor(aMillisecondTime/msPerHour) % HoursPerDay);
}

// Work out the minutes of the hour
function MinFromTime(aMillisecondTime)
{
   return (Math.floor(aMillisecondTime/msPerMinute) % MinutesPerHour);
}

// Work out the seconds of the minute
function SecFromTime(aMillisecondTime)
{
   return (Math.floor(aMillisecondTime/msPerSecond) % SecondsPerMinute);
}

// Work out the millisecond time value
function msFromTime(aMillisecondTime)
{
   return (aMillisecondTime % msPerSecond);
}
```

See also:	Broken down time, Calendar time, `Date` object, `MakeDate()`, `MakeDay()`, `MakeTime()`, Time range, Universal coordinated time

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.10

ECMA 262 edition 3 – section – 15.9.1.10

Time within day (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

Calculating the time within a day uses modulo arithmetic to find the remainder after calculating the day number.

The formula for calculating time within the day is shown here:

- t = an instant in time measured in milliseconds relative to 01-January-1970 UTC.
- msPerDay = 86400000
- TimeWithinDay(t) = t modulo msPerDay

Example code:

```
// Define some constants
HoursPerDay      =   24;
MinutesPerHour   =   60;
SecondsPerMinute =   60;
msPerSecond      = 1000;

// Derived values
msPerMinute = msPerSecond * SecondsPerMinute;
msPerHour   = msPerMinute * MinutesPerHour;
msPerDay    = msPerHour   * HoursPerDay;

// Grab the time now in milliseconds
myMilliseconds = Number(new Date());
document.write("Time within day ...: ");
document.write(TimeWithinDay(myMilliseconds));
document.write("<BR>");

// Work out the millisecond time value
function TimeWithinDay(aMillisecondTime)
{
   return (aMillisecondTime % msPerDay);
}
```

See also:	Broken down time, Day number, Time from year, Time range

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.2

ECMA 262 edition 3 – section – 15.9.1.2

TimeClip() (Time calculation)

A date and time algorithm.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

The `TimeClip()` operator is provided internally to convert implementation dependent numeric values to millisecond time values with a bounds check and sign handling fix up. Any date value is trimmed off and discarded, hence the name `TimeClip()`.

Implementations may represent number values internally in many ways. Time computations need the millisecond value to be presented in a particular way. This operator does all the cleaning and conversion necessary.

Although this is called an operator in the standard, its behavior is more like that of a function. It is not part of the formal language implementation and is probably not useful to have simulated in script form. It is documented in the standard to assist in the algorithmic breakdown of the `Date` method handlers and is covered here for the sake of completeness.

The result is a time value in milliseconds.

See also:	Broken down time, `Date` object, `Date()`, `Date.setTime()`, `Date.UTC()`

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.14

ECMA 262 edition 3 – section – 15.9.1.14

Timeout handlers (Definition)

You can set event handlers to be called when a timeout expires.

This is a mechanism for scheduling the execution of a script at some time in the future. The delay is measured from the moment the `setTimeout()` method is invoked.

See also:	`Window.clearInterval()`, `Window.clearTimeout()`, `Window.setInterval()`, `Window.setTimeout()`

Timer events (Definition)

Events that are triggered after a delay time.

Timer events are created by calling a timer set-up function and giving it a delay value. You can also cancel timeouts, but this can lead to problems because cancelling a timeout that has already been processed is prone to crashing the browser. Managing timeouts is tricky and exposes you to an area where the browser is somewhat less reliable than mainstream functionality.

Nevertheless, timer events are useful for managing refreshes and animation. You can interlock a refresh with some animation or scrolling so that the refresh exhibits as few screen redrawing artifacts as possible.

If you are really careful about memory leaks, you can build a ticker display with JavaScript. Making this clickable with an anchor is useful, but also exposes some cross-platform shortcomings and there are several ways to do this and each only works in one browser/platform combination and crashes on the others.

Because `setTimeout()` defines a delay from the time of execution to the time when the event is triggered, a cyclic mechanism takes no account of the processing time between `setTimeout()` calls. This means that the animation can become jerky. You can smooth that out by calculating how much time is left before you want the next scheduled event to occur. You need to access some kind of tick count to establish a reference time. You can do this by extracting values from a `Date` object and using its millisecond value. A modulo of that with the result subtracted from your ideal schedule frequency should yield a value that takes account of any processing jitter.

Automatic refreshing is sometimes called client pull.

If you want to execute something on a regular basis, then `setInterval()/clearInterval()` may be what you are looking for. These two methods manage an interval timer that you can set up to execute a function periodically. Because this doesn't disappear, there is less likelihood of crashing the browser if you try to clear one of these timers.

Warnings:

- Timer related crashes seem to be able to lock up a desktop system comprehensively enough that it requires a reboot to recover. This applies equally to Macintosh and Windows based systems and both MSIE and Netscape. There are many different causes, usually to do with runaway events, trying to kill timers that don't exist, or handling mouse rollovers that cross from one object to another. Multiple frames (especially `<IFRAME>` handling in MSIE on Macintosh) are especially prone to rollover crashes when combined with timer events. The crashing is all to do with timer events disposing of an object that the mouse has rolled over and triggered an interaction event for dispatch. In the meantime the object has gone and the browser follows it.

See also:	Event, Event handler, Event model, `Event` object, `Window.clearInterval()`, `Window.clearTimeout()`

Cross-references:

Wrox *Instant JavaScript* – page – 55

<TITLE> (HTML Tag)

A tag that encloses the title block in a document header. It corresponds to the MSIE `TITLE` object.

The `<TITLE>` tag conveys no apparent visible effect on the document. It is considered to be an invisible tag, although its value does appear in the window heading bar.

Warnings:

- Be careful not to confuse this with the `TITLE` attribute of an HTML tag and its corresponding `Element` object's `title` property. They are just a means of naming objects and associating specific tags with specific objects.

See also:	`Document.title`, `TITLE` object

TITLE object (Object/HTML)

An MSIE object that represents the `<TITLE>` block of a document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0

Inherits from:	`Element` object	
JavaScript syntax:	IE	*myTITLE* = document.all.tags("TITLE")[0]
	IE	*myTITLE* = document.all[*anIndex*]
	IE	*myTITLE* = *myDocument*.all.*anElementID*
	IE	*myTITLE* = *myDocument*.all.tags("TITLE")[*anIndex*]
	IE	*myTITLE* = *myDocument*.all[*aName*]
	-	*myTITLE* = *myDocument*.getElementById(*anElementID*)
	-	*myTITLE* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myTITLE* = *myDocument*.getElementsByTagName ("TITLE")[*anIndex*]
HTML syntax:	<TITLE> ... </TITLE>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`text`	
Event handlers:	`onClick`, `onDblClick`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`	

The <TITLE> tag conveys no apparent visible effect on the document. It is considered to be an invisible tag although its value does appear in the window heading bar.

See also:	<TITLE>, `Document.title`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`text`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

TITLE.text (Property)

The text contained inside the `<TITLE>` block for the current document.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTITLE*`.text`

This yields the text of the title block. This is the same as the `document.title` property. Because it is read-only, you cannot modify the value from script. You can't get at the content of the `<TITLE>` tag with the `innerHTML` property either.

See also:	`Document.title`

Property attributes:

`ReadOnly`.

ToBoolean (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This internal operator converts the public types to Boolean values.

The `ToBoolean` operator converts its argument to a value of type Boolean according to the following table:

Input Type	Result
Undefined	Always false.
Null	Always false.
Boolean	No conversion, the input value is returned unchanged.
Number	The result is `false` if the argument is 0 or `NaN` otherwise it is `true`.
String	Zero length strings return `false` otherwise the result is `true`.
Object	Always `true`.

See also:	`Cast` operator, Conversion, Implicit conversion, Logical operator, Number, Type conversion

Property attributes:

`Internal`.

Cross-references:

ECMA 262 edition 2 – section – 9.2

ECMA 262 edition 3 – section – 9.2

ToInt32 (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This internal operator converts its input value to a 32 bit signed integer value.

See also:	`Cast` operator, Conversion, Implicit conversion, Number, Type conversion

Property attributes:

`Internal`.

Cross-references:

ECMA 262 edition 2 – section – 9.5

ECMA 262 edition 3 – section – 9.5

ToInteger (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This internal operator converts values to integers.

See also:	`Cast` operator, Conversion, Implicit conversion, Type conversion

Property attributes:

`Internal`.

Cross-references:

ECMA 262 edition 2 – section – 9.4

ECMA 262 edition 3 – section – 9.4

Token (Definition)

Tokens are the fundamental components that an executable script is built from.

Availability:	ECMAScript edition – 2

A token is the smallest component of a source script text that can be interpreted.

Tokens are the actual components that an executable script is built from. They may be reserved words, identifiers, punctuator symbols or literals.

You cannot use reserved words as identifier names.

The reserved word set is comprised of the structural statements that you use to make a script work.

The `null` literal is simply the keyword `null`.

The Boolean literals are the keywords `true` and `false`.

See the reserved words topic for a list of all currently defined reserved words.

See also:	Boolean literal, `Cast` operator, Identifier, Keyword, Lexical convention, Lexical element, Literal, Null literal, Punctuator, Reserved word, Reserved Word

Cross-references:

ECMA 262 edition 2 – section – 7.4

ECMA 262 edition 3 – section – 7.5

ToNumber (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This internal operator converts its argument to an appropriate numeric value.

The `ToNumber` operator converts its input values according to the following table:

Input Type	Result
Undefined	Returns `NaN`
Null	0
Boolean	1 if `true`, 0 if `false`.
Number	No conversion, the input value is returned unchanged.
String	The value of a sequence of characters that can reasonably be converted to a number, and if not then `NaN` is returned.
Object	Internally, a conversion to one of the primitive types happens followed by a conversion from that type to a number. Some objects will return a number that is readily usable. Others will return something that cannot be converted and `NaN` will result.

The string scanning algorithm copes with spelled out special values such as Infinity, exponential values and can scan integers in Octal and Hexadecimal notation as well as decimal.

See also:	`Cast` operator, Conversion, Implicit conversion, `isFinite()`, `isNaN()`, Number, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 9.3

ECMA 262 edition 3 – section – 9.3

ToObject (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This internal operator converts its input argument into an object.

The `ToObject` operator converts its input arguments according to the following table:

Input Type	Result
Undefined	Generates a run-time error.
Null	Generates a run-time error.
Boolean	Create a new `Boolean` object whose default value is the input value.
Number	Create a new `Number` object whose default value is the input value.
String	Create a new `String` object whose default value is the input value.
Object	No conversion, the input value is returned unchanged.

See also:	`Cast` operator, Conversion, Implicit conversion, Number, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 9.9

ECMA 262 edition 3 – section – 9.9

http://cm.bell-labs.com/cm/cs/doc/90/4-10.ps.gz

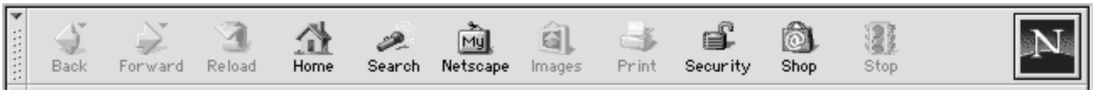
http://cm.bell-labs.com/netlib/fp/dtoa.c.gz

http://cm.bell-labs.com/netlib/fp/g_fmt.c.gz

toolbar (Property)

An alias for the `window.toolbar` property.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Bar object`	
JavaScript syntax:	-	*myWindow*`.toolbar`
	-	`toolbar`



See also:	`Bar` object, `Window.toolbar`

Property attributes:

`ReadOnly`.

top (Property)

An alias for the `window.top` property.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*myWindow*`.top`
	-	`top`

Property attributes:

`ReadOnly`.

Refer to:

`Window.top`

Topic classification (Overview)

Descriptions of topic classifications.

The various classifications of topics in this manual are listed in this table:

Classification	Description
Accessor method	A means of accessing properties
Additive operator	An operator that sums values
Arithmetic operator	An operator that works with numeric values
`Array`	A kind of object that is compounded from several others
ASP tag	A tag that is only available in Microsoft ASP
Assignment operator	An operator that assigns values
Attribute	Attributes define how properties behave
Background	Background information
Bitwise operator	An operator that works with bit patterns
Blend filter	A transition filter available only in MSIE
Boolean literal	A Boolean constant value
Browser object	An object only available in a browser
Built-in function	A function provided by the implementation
Built-in object	An object provided by the implementation
Collection	A sub-class of the `Array` object that supports additional methods
Comma operator	An expression delimiter
Conditional operator	An operator that yields a Boolean result
Constant	A fixed literal value
Constructor	An object instance factory
Core object	An object that is part of the core language
Declaration	A specification of an identifier
Definition	A specification of an identifier with storage
Delimiter	A token separator
DOM object	An object that belongs to the DOM specification
Environment variable	Part of the implementation that an interpreter runs in
Escape sequence	A means of generating non-typable characters
Event	Events need to be considered a separate kind of language entity

Table continued on following page

Classification	Description
Event handler	Event handlers are many and various, each covered in its own topic
External code call	Some environments support the calling of external code
File extension	File types are identified by different file extensions
`Function` object attribute	An attribute of a `function` object
Function property	An object property that can be called as a method
Global function	A function that belongs to the `global` object
Global variable	Some variables are available as member properties of the `global` object
Host object	Objects not defined as part of the core language
HTML tag	A part of an HTML document
HTML Tag	Some JavaScript language needs to be described in terms of its underlying HTML tags
HTML Tag Attribute	An attribute added to an HTML tag
Identifier	The name of a function or variable
Idiom	A familiar form or expression
Instance	A copy of a built-in object
Internal function	A function that cannot be called from a script
Internal method	A method that cannot be invoked from a script
Internal object	An object maintained internally by the interpreter and not generally available to the script writer
Internal object	An object that cannot be accessed from a script
Internal operator	An operator that cannot be used in expressions contained in a script
Internal property	A property that cannot be accessed from a script
Internal type	A data type that cannot be created from a script
Interpreter	There is now a wide choice of JavaScript interpreters available
Iterator	Certain control structures that can be used to create loops
Java class	Access to `Java` objects is via the wrappers that let you access the classes
Java method	A method belonging to a `Java` object accessible via a JavaScript wrapper
Java package	A collection of Java classes enclosed in a JavaScript wrapper
Java static method	A method belonging to a Java class accessible via a JavaScript wrapper
`JScript` object	An object only available in Microsoft JScript
Keyword	A reserved word in the language
Label	Code can be labelled with entry points
Logical operator	An operator that works with Boolean values
Method	A message sent to a method performs an action inside the object
MIME type	A means of telling the browser what kind of data is being supplied
Mobile communicator	Web access is also possible from mobile devices
MSIE method	A method that is only available in MSIE even though the object may be more widely available

Table continued on following page

Classification	Description
MSIE object	An object only available in MSIE
MSIE property	A property that is only available in MSIE even though the object may be more widely available
Native object	A built-in object defined by the standard
NES result value	A result value that a Netscape Enterprise Server method call returns
NES server method	A method that is only available in Netscape Enterprise Server even though the object may be more widely available
NES server object	An object only available in Netscape Enterprise Server
NES server property	A property that is only available in Netscape Enterprise Server even though the object may be more widely available
Netscape object	An object that is only available in the Netscape scripting environment
Null literal	A `null` constant value
Object	An object of no particular type
Object	An object that can be instantiated
Object model	There are several object models that describe various parts of the JavaScript object space
Operator	An operator that can be used to create expressions
Overview	A descriptive outline text
Pitfall	Some topics describe ways in which JavaScript can catch you out
Portable Documents	JavaScript is used to handle forms in PDF
Postfix operator	An operator that can be appended to an LValue
Pre-processor directive	The Microsoft interpreters support a pre-processor rather like that in the C language
Primitive value	A fundamental value built-in to the implementation
Product	Some JavaScript interpreters are available as part of a commercial product
Property	A named attribute of an object
Proxy.pac support	Part of the Netscape proxy handler
References	Sources of additional information
Relational operator	An operator that yields the result of a comparison
Request method	Several types of request method can be used in a URL
Reserved word	Reserved functionality
Reveal filter	A transition filter available only in MSIE
Script container	Scripts may be carried in a variety of containers other than HTML documents
Security model	A particular way of implementing security
Security policy	A policy regarding the way security is managed
Security privilege	A privilege level in the security mechanism
Selector	A means of choosing one of several possible outcomes

Table continued on following page

Classification	Description
Server object	An object that is only available server-side
Server-side file	Special files accessible only to the web server
Server-side method	Some methods are available only on the server-side
Server-side object	Some objects are only available on the server-side
Simulated functionality	Some script that emulates capabilities of other languages
Special file	JavaScript is implemented in a variety of ways and in some cases special files are necessary
Standard	Several standards affect the way that JavaScript works
Statement	A single executable unit of code
Static method	A method belonging to a class, not an object
Static property	These properties belong to the class (or rather they would in a truly object oriented system)
String operator	An operator for working with string data
Summary	A brief description of a subject
Support for preferences	Preference handling requires a number of different topics to describe it
Symbol	A special character
Time calculation	A process of working out time values
Time operator	An operator for time value expressions
Transition filter	A transition filter available only in MSIE
TV Set-top Box	Web access is now becoming more widely available in TV applications
Type	A data value classification
Unary operator	An operator that only requires one operand
Useful tip	Some topics provide useful advice
Value	A value of a variable or object
Value property	A property of an object that yields a value and cannot be called as a function
Visual filter	A transition filter available only in MSIE
Web browser	Each of the commonplace web browsers is described in its own topic
Web server	Server-side code needs to be executed under control of a web server

See also:	ECMA

ToPrimitive (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This operator converts objects to primitive values.

The `ToPrimitive` operator takes a value argument and an optional preferred type argument and converts its input to a primitive type from an object representation.

Input Type	Result
Undefined	No conversion, the input value is returned unchanged.
Null	No conversion, the input value is returned unchanged.
Boolean	No conversion, the input value is returned unchanged.
Number	No conversion, the input value is returned unchanged.
String	No conversion, the input value is returned unchanged.
Object	The default value defined by the object's internal `DefaultValue` method is returned. A coercion to the preferred type happens and is context dependent on where the result is being assigned.

See also:	`Cast` operator, Conversion, Implicit conversion, Number, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 8.6.2.6

ECMA 262 edition 2 – section – 9.1

ECMA 262 edition 3 – section – 8.6.2.6

ECMA 262 edition 3 – section – 9.1

ToString (Operator/internal)

Returns a string primitive version of an object.

Availability:	ECMAScript edition – 2

This internal operator converts its input argument to a string.

The `ToString` operator converts its input arguments according to the following table:

Input Type	Result
Undefined	"`undefined`"
Null	"`null`"
Boolean	If the argument is true, then the result is "`true`" otherwise the result is "`false`"
Number	Special cases are provided for NaN and Infinity where "`NaN`" and "`Infinity`" will be returned. Otherwise the string is a textual representation of the value. The string is formatted into decimal or exponential formats as determined by the magnitude of the value.

Table continued on following page

Input Type	Result
String	No conversion, the input value is returned unchanged.
Object	An internal conversion to a primitive takes place followed by a conversion from that primitive to a string. Some objects will return a string value that is immediately useful.

Warnings:

- In Microsoft environments, this is available most of the time but does not work for certain objects. In particular, there may some objects in WSH for which it is not supported.

See also:	`Cast` operator, Conversion, Implicit conversion, `JSObject.toString()`, String concatenate (+), `toString()`, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 9.8

ECMA 262 edition 3 – section – 9.8

toString() (Function/global)

Returns a string representation of the receiving object.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myObject*`.toString()`
	-	*myObject*`.toString(`*aRadix*`)`
Argument list:	*aRadix*	Radix conversion can be applied when the receiver is a number

The generic behavior of this method is to return a String primitive representation of the receiving object. It is generally overridden on a class by class basis due to objects containing such different properties and values.

The `ToString` internal operator is called. This doesn't usually tell you very much. The default `toString()` handlers may be different for the built-in classes, but all you'll likely get from a class you create yourself will be the string "`[object Object]`".

You will need to override the `toString()` function that is provided by default and add your own. This should be added to the prototype of your class.

The generic version of the `toString()` method may be useful when debugging. You can use the `apply()` method to force its use on objects you are trying to inspect and which may have overridden the `toString()` method themselves.

This method is supported by virtually every object by virtue of the fact that it is available as a method of the `Global` object in Netscape. Therefore it gets inherited into the scope chain for every script and function (method).

Warnings:

- At JavaScript version 1.2 in the Netscape version 4 browser, there is a slight difference in the way that `toString()` works. It will output all the nested objects that are joined by properties. This gave rise to a technique for deep copying objects. However, it wasn't ECMA compliant and it no longer works in Netscape as of JavaScript 1.3.
- It might still work if you set the language version to JavaScript 1.2, but it's unreliable, not portable and definitely not going to work in Netscape 6.0. If you are exploiting it, you need to find an alternative because your scripts are going to break.

See also:	`Array.toString()`, `Boolean.toString()`, Conversion, `Date.toString()`, `Error.toString()`, `Function.toString()`, `Number.toString()`, `Object.toString()`, `prototype.toString()`, `RegExp.toString()`, String, String concatenate (+), `String.toString()`, `ToString`

ToUint16 (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This internal operator converts its input argument into an unsigned 16 bit integer value.

See also:	`Cast` operator, Conversion, Implicit conversion, Number, `String.fromCharCode()`, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 9.7

ECMA 262 edition 3 – section – 9.7

ToUint32 (Operator/internal)

An internal operator for converting values.

Availability:	ECMAScript edition – 2

This internal operator converts its input argument to an unsigned 32 bit integer.

See also:	`Cast` operator, Conversion, Implicit conversion, Number, Type conversion

Cross-references:

ECMA 262 edition 2 – section – 9.6

ECMA 262 edition 3 – section – 9.6

TR object (Object/HTML)

An object that encapsulates the row content of a table contained in a `<TR>` tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myTR* = *myDocument*.all.*anElementID*
	IE	*myTR* = *myDocument*.all.tags("TR")[*anIndex*]
	IE	*myTR* = *myDocument*.all[*aName*]
	-	*myTR* = *myDocument*.getElementById(*anElementID*)
	-	*myTR* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myTR* = *myDocument*.getElementsByTagName("TR")[*anIndex*]
HTML syntax:	`<TR> ... </TR>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`align`, `bgColor`, `borderColor`, `borderColorDark`, `borderColorLight`, `ch`, `chOff`, `rowIndex`, `sectionRowIndex`, `vAlign`	
Object methods:	`deleteCell()`, `insertCell()`	
Event handlers:	`onBlur`, `onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	
Collections:	`cells[]`	

Tables are a two dimensional array of cells divided first into rows and then into columns. To access a particular cell, you locate its row first and then index along the cells in a row to find the one you want.

Rows are enclosed in `<TR>` tags which instantiate a `TR` object. The `TR` objects in a table are available as members of the `rows[]` collection.

They are also available in the `rows[]` collections that belong to `TFOOT`, `TBODY` and `THEAD` objects if the rows have been grouped inside tags that instantiate those objects.

You can access the cells in a row by using the `cells[]` collection that belongs to the `TR` object for the row you are interested in.

Warnings:

- Some earlier versions of the MSIE browser did not support access to the `innerHTML` property and related content when used on the Macintosh platform.

See also:	`Element` object, `TABLE` object, `TABLE.deleteRow()`, `TABLE.insertRow()`, `TBODY` object, `TD` object, `TFOOT` object, `THEAD` object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`align`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`bgColor`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`borderColor`	-	3.0 +	-	4.0 +	-	-	-	-
`borderColorDark`	-	3.0 +	-	4.0 +	-	-	-	-
`borderColorLight`	-	3.0 +	-	4.0 +	-	-	-	-
`ch`	1.5 +	-	6.0 +	-	-	1 +	-	-
`chOff`	1.5 +	-	6.0 +	-	-	1 +	-	-
`rowIndex`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
`sectionRowIndex`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	ReadOnly
`vAlign`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`deleteCell()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`insertCell()`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onBlur`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	-	Warning
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

TR.align (Property)

The alignment of content within table cells in the row.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTR*`.align`

The alignment of the `TR` object with respect to its containing parent object is defined in this property. The following set of alignment specifiers are available:

- `center`
- `left`
- `right`
- `char`
- `justify`

See also:	`TABLE.align`, `TBODY.align`, `TD.align`, `TFOOT.align`, `TH.align`, `THEAD.align`

TR.bgColor (Property)

The background color of table cells within the row.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTR*`.bgColor`

The background can be colored independently of whether an image is loaded into the background of an object. In fact it may be advisable to set the background color to something similar to the average color of the background image in case the image takes a long time to load or the browser is unable to display a background image.

See also:	Color names, Color value

TR.borderColor (Property)

The color of the border around table cells in the row.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTR*`.borderColor`

You can use this property to determine the color of a border surrounding a table row. Note that there are additional properties to determine the highlights and lowlights of the table row border coloring.

TR.borderColorDark (Property)

The color value of the shadowed edge of the table row border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTR*`.borderColorDark`

Table row borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the shadowed part of the table row border.

TR.borderColorLight (Property)

The color value of the highlighted edge of the table cell border (assuming the table is lit from the top left).

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myTR*`.borderColorLight`

Table row borders are presented as having an engraved appearance. This means you may need to control the highlights and lowlights. This property defines the color of the highlighted part of the table row border.

TR.cells[] (Collection)

A collection of cells within this row of the table.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	-	*myTR*`.cells`

The `table.cols` value should also be equivalent to `table.rows.cells.length` value.

See also:	`TABLE.cols`

Property attributes:

`ReadOnly`.

TR.ch (Property)

The alignment character for cells in a column arrangement.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	*myTR*`.ch`

HTML 4.0 provides for text to be arranged in neat columns within table cells without the need to create additional tables within tables. This method of alignment is selected by setting the `ALIGN="CHAR"` HTML tag attribute. The `CHAR` HTML tag attribute is reflected in this property and is active when the `CHAROFF` HTML tag attribute is present.

See also:	`COL.ch`, `COLGROUP.ch`, `TD.ch`, `TH.ch`, `THEAD.ch`

TR.chOff (Property)

The offset of a column alignment character.

Availability:	DOM level – 1 JavaScript – 1.5 Netscape – 6.0
Property/method value type:	`String primitive`

JavaScript syntax:	N	*myTR*.chOff

The CHAR alignment style requires that an alignment character be specified with the `ch` property and that an offset measured in pixels be defined as its value. The offset value can be defined with the `CHAROFF` HTML tag attribute. The remainder of the string is offset by this distance from the alignment character.

See also:	`COL.chOff`, `COLGROUP.chOff`, `TD.chOff`, `TH.chOff`, `THEAD.chOff`

TR.deleteCell() (Method)

In a multiple column table, you can delete a particular cell within a row with this method.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
JavaScript syntax:	-	*myTR*.deleteCell(*anIndex*)
Argument list:	*anIndex*	The horizontal cell address to be deleted

See also:	`THEAD.deleteRow()`

TR.insertCell() (Method)

You can insert an additional cell into the row within a table with this method.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`TD object`	
JavaScript syntax:	-	*myTR*.insertCell(*anIndex*)
Argument list:	*anIndex*	The horizontal index at which to insert a cell

TR.rowIndex (Property)

A zero-based integer that indicates which row of the table this <TR> block represents.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTR*`.rowIndex`

The `TR.rowIndex` property is the vertical coordinate for cells in this row measured across the entire table.

This property yields a zero-based index number for this table row within the `rows[]` collection belonging to the `TABLE` object the row is a member of.

Note that this index will not hold true for the `rows[]` collections belonging to `THEAD`, `TBODY` and `TFOOT` objects. For that, you should inspect the `sectionRowIndex` property.

See also:	`TABLE.cells[],TABLE.rows[],TBODY.rows[], TFOOT.rows[],THEAD.rows[],TR.sectionRowIndex`

Property attributes:

`ReadOnly`.

TR.sectionRowIndex (Property)

A zero-based integer that indicates which row of the table this <TR> block represents.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myTR*`.sectionRowIndex`

The `TR.sectionRowIndex` property is the vertical coordinate for cells in this row measured within a `THEAD`, `TBODY` or `TFOOT` group and is reset to zero at the start of each section of the table.

This property yields a zero-based index number for this table row within the `rows[]` collection belonging to the `THEAD`, `TBODY` or `TFOOT` object the row is a member of.

Note that this index will not hold true for the `rows[]` collections belonging to `TABLE` object. For that, you should inspect the `rowIndex` property.

See also:	`TR.rowIndex`

Property attributes:

`ReadOnly`.

TR.vAlign (Property)

The vertical alignment of content in the table cells in this row.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myTR*`.vAlign`

The vertical alignment of the content within the table cells can be controlled across an entire `TR` extent with this property. The following keywords can be assigned to it:

- `baseline`
- `bottom`
- `middle`
- `top`

These may be also available on some implementations:

- `absbottom`
- `absmiddle`
- `baseline`
- `texttop`

See also:	`TBODY.vAlign`, `TD.vAlign`, `TFOOT.vAlign`, `TH.vAlign`, `THEAD.vAlign`

transient (Reserved word)

Reserved for future language enhancements.

Refer to:

Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

Transition (Definition)

A visual effect used during event handling in MSIE.

These are activated by means of the `apply()` and `play()` methods. When executed, the transition will reveal the new display content. This mechanism obviously requires the use of a second display buffer and so for large onscreen windows, you may find that MSIE requires much more memory to utilize these effects properly.

See also:	`Element.filters[]`, filter – `BlendTrans()`, filter – `RevealTrans()`, `Filter` object, `style.filter`

Translation (Definition)

The process of tokenizing and interpreting a script.

See also:	Error handling, Interpret

Trigonometric function (Definition)

Functions for calculating angular values.

The trigonometric functions are provided by the `Math` object.

The following trigonometric functions are defined by the ECMA standard:

Function	Description
`acos()`	Inverse cosine
`asin()`	Inverse sine
`atan()`	Inverse tangent
`atan2()`	Inverse tangent of two values
`cos()`	Cosine of an angle
`sin()`	Sine of an angle
`tan()`	Tangent of an angle

See also:	Exponent-log function, Integer-value-remainder, `Math` object, `Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`, `Math.tan()`, Mathematics, Power function

true (Primitive value)

The Boolean `true` value.

Availability:	ECMAScript edition – 2
Property/method value type:	`Boolean primitive`

This is a Boolean primitive value representing the logically true state.

Conditional code execution depends on this value to signify the execution of a block of script code.

See also:	Boolean, Boolean, Boolean literal, Definition, false

Cross-references:

ECMA 262 edition 2 – section – 9.2

ECMA 262 edition 2 – section – 15.6

ECMA 262 edition 3 – section – 9.2

ECMA 262 edition 3 – section – 15.6

try ... catch ... finally (Statement)

A mechanism for attempting to execute some potentially problematic code with a means of catching the exception and continuing execution.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape – 6.0

This was introduced in JavaScript 1.4 and JScript version 5.0. The intention is to provide a way to execute a section of code in the `try` block and then if it has a problem, some recovery action can trap the error and handle it gracefully.

There are three basic sections.

The `try` statement is followed by a block of code enclosed in braces that may be problematic. Indeed, the problems may not be script-based errors but may be the result of testing for some condition. That may lead to a custom exception being thrown.

If an exception of any kind happens in the `try` block, execution is immediately passed to the `catch()` function following.

The `catch` function is passed an `Error` object containing details of the kind of exception that has occurred.

When the `catch` function completes, execution drops into the block of code associated with the `finally` statement. In the case of the `try` block not having any exceptional behavior, at the end of that block execution also drops into the `finally` code block, bypassing the `catch` function altogether. So the `finally` code gets executed always after the `try` block.

You might use the `finally` block to tidy up or discard some unwanted objects.

In the example, the `try` block makes sure two values are presented in the correct order. If they are not in the right order, an exception is thrown and during the exception handling, they are swapped over. We put up an alert and set a flag that is presented in the output just to be sure it really happened like that.

Warnings:

- The functionality is not supported in Netscape prior to version 6.0.
- This is not supported by Netscape Enterprise Server 3 and so its error handling capabilities are not available server-side.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
testThrow(100, 200);
testThrow(300, 100);
function testThrow(arg1, arg2)
{
   var switched = "NO";

   // Force an error condition
   try
   {
      if(arg1 < arg2)
      {
         throw "Wrong order";
      }
   }
   catch(myErr)
   {
      alert(myErr);
      var temp = arg1;
      arg1 = arg2;
      arg2 = temp;
      switched = "YES"
   }
   finally
   {
```

```
        document.write("Biggest : "  + arg1 + "<BR>");
        document.write("Smallest : " + arg2 + "<BR>");
        document.write("Switched : " + switched + "<BR>");
        document.write("<BR>");
    }
}
</SCRIPT>
</BODY>
</HTML>
```

See also:	`catch( ... )`, `Error` object, `EvalError` object, Exception handling, `finally ...`, `RangeError` object, `ReferenceError` object, `SyntaxError` object, `throw`, `TypeError` object, `URIError` object

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

ECMA 262 edition 3 – section – 12.14

TT object (Object/HTML)

An object that represents the font style controlled by the `<TT>` HTML tag.

Availability:	JScript – 3.0 Internet Explorer – 4.0 Deprecated	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myTT* = *myDocument*.all.*anElementID*
	IE	*myTT* = *myDocument*.all.tags("TT")[*anIndex*]
	IE	*myTT* = *myDocument*.all[*aName*]
	-	*myTT* = *myDocument*.getElementById(*anElementID*)
	-	*myTT* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myTT* = *myDocument*.getElementsByTagName("TT")[*anIndex*]
HTML syntax:	`<TT> ... </TT>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	`onClick`, `onDblClick`, `onDragStart`, `onFilterChange`, `onHelp`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onSelectStart`	

`<TT>` tags and the objects that represent them are inline elements. Placing them into a document does not create a line break.

See also:	`Element` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onDblClick`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyPress`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onKeyUp`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseDown`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseMove`	-	3.0 +	-	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseOver`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onMouseUp`	-	3.0 +	-	4.0 +	3.0 +	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

TV Set-top boxes (Definition)

The integration of TV and Web content is resulting in new kinds of hardware that displays web page content at TV resolution.

In the same way that we had a browser war, there is a similar activity going on with TV set-top boxes. These are generally a slightly lower than normal functionality browser burned into a ROM or upgradeable Flash PROM chip.

There are several varieties already available in various forms. From the MSIE tradition comes the WebTV box. This is a browser originally based on MSIE 3 but now enhanced somewhat. There are limitations due to the need for it to occupy as little memory as possible. For those who prefer the Netscape browser, the Liberate box comes from a company initially formed as a joint venture between Netscape Communications and Oracle. It now has significant backing from Cisco and all things considered may well become the dominant platform.

Other alternatives exist from Bush Electronics and Netgem.

Because of the hardware limitations, these 'browser in a box' systems tend to support a reduced functionality set when compared with the more recent desktop computer based browsers. Typically the HTML support would be at version 3.2 of HTML and if JavaScript is supported it would be based on JavaScript version 1.2 with some minor additions and a few features omitted or compromised. This is very much a generalization however, and you will likely find some products that exceed these specifications.

A good source of information about the progress and development of the whole area of Digital Interactive TV and set-top boxes can be found at the Ruel web site.

See also:	JellyScript, Liberate TV Navigator, OpenTV, WebTV

Web-references:

http://ruel.net/top.box.news/

Type (Definition)

The ECMA 262 standard defines nine data types. A type is a set of data values.

Availability:	ECMAScript edition – 2

A value is an entity that can take on the personality of one of the implemented data types. Some types are reserved for internal use and are not generally made visible to the script developer although this may be implementation dependent and they might be visible with a debugger.

The ECMA 262 standard defines nine data types. The host environment may add others depending on what it needs. The various types are listed in the table, which indicates whether they are special internal types:

Type Name	Description
Aggregate	A collection of atomic types assembled collectively into an object.
Arithmetic	All types that yield a value that can be operated on numerically.
Array	Collections of objects and identifiers assembled into a sequence.
Basic	The fundamental simple, non-object types.
Boolean	This type can store and yield `true` or `false` values.
Completion	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
List	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
Null	This has exactly one value, `null`, and is distinct from undefined.
Number	Integer and floating point values are all stored in a generic number type.
Object	An object is an unordered collection of properties. Each property consists of a name, a value and a set of attributes.
Reference	Used only as the intermediate result of expression evaluations and cannot be stored in object properties.
Scalar	The non-object types.
String	Strings are arrays of characters which are accessible individually by indexing their position in the sequence.
Undefined	This value is returned by variables that have not yet been assigned a value.

Warnings:

- You cannot access values of type Reference, List and Completion from your scripts and they cannot be stored as object properties.

See also:	Aggregate type, Boolean, `Cast` operator, Completion type, `const`, `enum`, Identifier, `int`, List type, Logical operator, `null`, Number, Object, Reference, Scalar type, String, undefined type

Cross-references:

ECMA 262 edition 2 – section – 4.3.1

ECMA 262 edition 2 – section – 8

ECMA 262 edition 3 – section – 4.3.1

ECMA 262 edition 3 – section – 8

Type conversion (Definition)

Type conversion happens automatically as needed.

Availability:	ECMAScript edition – 2

JavaScript on the one hand is very good for the beginning programmer because it is very forgiving in regard to variable types. Other languages require that you define the type of a variable at the outset and only ever store the correct kind of data in it. JavaScript doesn't actually care and modifies the type of a variable on the fly according to the values being presented.

Type conversion happens automatically as needed. To support this, a set of polymorphic operators are provided to make the conversion.

The following internal operators are provided:

- `ToPrimitive`
- `ToBoolean`
- `ToNumber`
- `ToInteger`
- `ToInt32`
- `ToUint32`
- `ToUint16`
- `ToString`
- `ToObject`

From the script writers point of view, these functions are inaccessible. However, you can call the constructors to invoke these functions indirectly.

The items of data themselves are still strongly typed, but the variable containers are smart enough to cope. There are some subtle rules to this however and you can create some situations where a very hard to diagnose problem occurs that is based on data typing inside the variables.

Type conversion happens during expression evaluation. You can still get rounding errors with Unary operators but you only need to worry about converting a single type.

If you think there is some ambiguity in your expression, you should force a conversion to the most appropriate type. Generally, object values should be converted to Numbers or Strings. Objects will generally have a preferred type that they yield when converted. In most cases that will be a number although `Date` objects prefer to become Strings.

Binary operators require two values and some require that both are of the same type or can be converted to the same type. If the type of the two operands is different, then you can have problems with some operators in expressions. If both need to be numeric for example, and once is not a legal numeric value even after conversion, then an error will result. The arithmetic and bitwise operators fall into that category.

The rest of the binary operators are polymorphic. That means they can cope with values of different types. The assignment operators will change the LValue to whatever type the RValue is. There is still some requirement here that the LValue is legally convertible. The comma operator doesn't combine its two operands in any way whatsoever.

Occasionally the interpreter may generate an error message due to its inability to legally convert a value to an appropriate type to complete the expression evaluation.

The relational operators cause some confusion since the values need to be converted for comparison but the relational comparisons work for several types. It is difficult to know what type will be used for the relational test unless you explicitly force a type conversion yourself.

The concatenation operator is also ambiguous because you may have two numeric strings and want to add them but the concatenate operator may join them together to make a string composed of both.

The addition/concatenation operator looks at the arguments and if either is a String already or preferentially converts to one, then a concatenation occurs. If neither operator prefers to be a String then a Number conversion happens and the values are added.

Relational operators perform the same deductive test. However, the conversion to Strings is less likely since relational operators are concerned with magnitude. However, they will test for collation order if both arguments are Strings.

Relational operators will attempt to convert both arguments to a Number and if they cannot, then they will use a String based compare. If at least one argument can be converted to a Number then the other will be forced to be a Number for comparison purposes. So for a relation test to be String based, both arguments must be Strings, or string-like objects.

Tests for equality require further deductive reasoning on the part of the interpreter. The values are converted to their preferred types. If the types are the same then the values can be compared easily either as Numbers or Strings. If the types are different, then further conversion is necessary before the comparison can be completed. In that case, Boolean values become Numbers as do any other non-Numeric values and Numeric comparison predominates.

Comparing `null` with undefined values does not require any conversion and they will compare equal.

Warnings:

- The ECMAScript compliant interpreters are based on comparison rules that prevailed prior to ECMAScript being formalized. However, JavaScript version 1.2 incorrectly anticipated that the standard would rule that equality required the type of the two operators to be identical. This means that a number in quotes will not test equal to the numeric value itself. Therefore "10" != 10 yields a `true` value instead of a `false` value. This will only come into play if you specify JavaScript version 1.2 in the language attribute of the `<SCRIPT>` tag in your web page. Placing a `<SCRIPT LANGUAGE=JavaScript">` tag is sufficient to work around this. In Netscape version 4.0, if you don't explicitly ask for version 1.2 of JavaScript, you still have access to the JavaScript version 1.2 capabilities of the language except that this particular comparison works as per the pre-JavaScript version 1.2 rules. That is to say, it works correctly.
- Version 6 of Netscape and version 5 of MSIE claim to be ECMAScript compliant and this problem should go away. Other browsers strive for ECMA compliance too, so as long as you avoid specifying JavaScript version 1.2 in your `<SCRIPT>` tags, you shouldn't encounter this problem.
- New operators `===` and `!==` are introduced with the ECMA standard at edition 3 to provide a test that takes account of data typing. MSIE version 4.0 already supports them in anticipation of the third edition of the standard. Netscape supports them at version 6.0.
- Earlier versions of MSIE and Netscape exhibited bugs in the comparison logic where the results of comparisons involving `NaN`, `null` and 0 where type conversions led to inconsistent behavior.

Example code:

```
// String object scanned from numeric literal
a = String(50);          // Equivalent to "50"

// Number object scanned from string literal
b = Number("22.22");     // Equivalent to 22.22

// String literal coerced to numeric
c = +("0x00FF");         // Equivalent to 255

// String with number subtracted
d = "22.22" - 11.11;     // Equivalent to 11.11

// Logical operator on string and number
e = ("abcde" && 23);     // Equivalent to 23

// Bitwise operator on string and number
f = ("0xF0" & 255);      // Equivalent to 240

// Number coerced to string by concatenation
g = "100" + 10;          // Equivalent to "10010"

// Test for equality of value
h = ("100.1" == 100.1);  // Equivalent to true

// Test for identity of instantiation
i = ("100.1" === 100.1); // Equivalent to false

// Object coerced to string forcing number to be concatenated
j = (new Object) + 10;   // Equivalent to "[object Object]10"
```

See also:	Add (+), Arithmetic operator, Bitwise operator, `Boolean()`, `Cast` operator, Equal to (==), Equality expression, `JSObject.toString()`, Logical operator, `Math` object, `Math.abs()`, `Math.ceil()`, `Math.floor()`, `Math.round()`, Minima-maxima, Number, Relational expression, String, String concatenate (+), `ToBoolean`, `ToInt32`, `ToInteger`, `ToNumber`, `ToObject`, `ToPrimitive`, `ToString`, `ToUint16`, `ToUint32`

Cross-references:

ECMA 262 edition 2 – section – 9

ECMA 262 edition 3 – section – 9

Wrox *Instant JavaScript* – page – 35

Wrox *Instant JavaScript* – page – 37

TypeError object (Object/core)

A native error object based on the `Error` object.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 Netscape – 6.0	
Inherits from:	`Error object`	
JavaScript syntax:	N	`myError = new TypeError()`
	N	`myError = new TypeError(aNumber)`
	N	`myError = new TypeError(aNumber, aText)`
Argument list:	*aNumber*	An error number
	aText	A text describing the error

This sub-class of the `Error` object is used when an exception is caused by an operand having an unexpected data type.

See also:	`catch( ... )`, `Error` object, `EvalError` object, `RangeError` object, `ReferenceError` object, `SyntaxError` object, `throw`, `try ... catch ... finally`, `URIError` object

Inheritance chain:

`Error` object

Cross-references:

ECMA 262 edition 3 – section – 15.1.4.14

ECMA 262 edition 3 – section – 15.11.6.5

typeof (Operator/unary)

An operator that yields the type of an operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 1.0 Internet Explorer – 3.02 Netscape – 3.0 Opera – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`typeof` *`anOperand`*
	-	`typeof(anOperand)`
Argument list:	*`anOperand`*	An object or variable to check for type

This operator produces a string that contains the operand's type.

The `typeof` operator inspects the operand and returns a string representing its type. The operand is not evaluated. There are times when this is advantageous and can avoid a run-time error.

The string value returned depends on the operand type being evaluated. The `typeof` operator returns these values:

Type	Result
Undefined	"undefined"
Infinity	"number"
NaN	"number"
Null	"object"
Boolean primitive	"boolean"
Number primitive	"number"
String primitive	"string"
`Boolean()` constructor	"boolean"
`Date()` constructor	"string"
`Number()` constructor	"number"
`RegExp()` constructor	"undefined"
`String()` constructor	"string"
`Boolean` object instance	"object"
`Date` object instance	"object"
`Math` object instance	"object"
`Number` object instance	"object"
`RegExp` object instance	"object"

Table continued on following page

Type	Result
`String` object instance	"object"
Generic object instance	"object"
Object not supporting a call interface	"object"
Object that supports a call interface	"function"
Other host objects	Implementation defined
`typeof` any value	"string"

Note that the values returned are lowercase and are not an exact match for the class of the operand. String objects and primitive strings are both described as having a `typeof` "string" for instance.

In some documentation the `typeof` operator is referred to as a function. Since it has optional parentheses and the operand is passed as an argument, it behaves as if it were a function.

The `typeof` operator is one of few ways in which you can use the contents of a variable that is not yet defined. It will yield the undefined type for variables that have not yet had a value assigned to them. And thus you can determine if it is safe to use them as an RValue in an assignment expression.

The associativity is right to left.

Refer to the operator precedence topic for details of execution order.

In JavaScript version 1.1, you can determine the difference between the undefined and `null` values.

In JavaScript version 1.3, the `===` operator will detect the difference.

You cannot distinguish between different kinds of objects. To do that you can compare the object `constructor` property with one of the built-in types. You can ask the constructor property (if there is one) for its name. Sometimes a `toString()` conversion on the object will yield a function name and occasionally you may need to check prototype values or test for the existence of properties. If all else fails, you have to assume that it's just an object of arbitrary type.

Warnings:

- The `typeof` operator is not available in Netscape version 2.02.
- In Microsoft environments, this is available most of the time, but does not work for certain objects. In particular, there may some objects in WSH for which it is not supported.

Example code:

```
// Testing a string value
var aString = 'String text';
document.write(typeof(aString));     // Yields "string"

// Testing variables that exist but are not yet assigned
var aVar1;
document.write(typeof aVar1);     // Yields "undefined"

// Testing variables that do not yet exist
document.write(typeof aVar2 );     // Yields "undefined"
```

See also:	Associativity, `Cast` operator, class, Enquiry functions, Equal to (==), `Global` object, Grouping operator ( ), Identically equal to (===), NOT Equal to (!=), NOT Identically equal to (!==), Object inspector, Operator, Operator Precedence, Reference, Special type, Unary expression, Unary operator, void

Cross-references:

ECMA 262 edition 2 – section – 11.1.4

ECMA 262 edition 2 – section – 11.4.3

ECMA 262 edition 3 – section – 11.4.3

O'Reilly *JavaScript Definitive Guide* – page – 47

Wrox *Instant JavaScript* – page – 21

Wrox *Instant JavaScript* – page – 22

U object (Object/HTML)

An object that represents the font style controlled by the <U> HTML tag.

Availability:	DOM level –1 JavaScript –1.5 JScript –3.0 Internet Explorer –4.0 Netscape version –6.0 Deprecated	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myU* = *myDocument*.all.*anElementID*
	IE	*myU* = *myDocument*.all.tags("U")[*anIndex*]
	IE	*myU* = *myDocument*.all[*aName*]
	-	*myU* = *myDocument*.*getElementById*(*anElementID*)
	-	*myU* = *myDocument*.getElementsByName(*aName*)[*anIndex*]
	-	*myU* = *myDocument*.getElementsByTagName ("U")[*anIndex*]
HTML syntax:	<U> ... </U>	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

<U> tags and the objects that represent them are in-line elements. Placing them into a document does not create a line break.

See also:	`Element` object

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

UDI (Definition)

Universal Document Identifier.

This is another name for a URL. Arguably this is a sub-set of all the available URLs because it refers specifically to a document and not a sound effect for example.

See also:	URI, URL, URN

UIEvent object (Object/DOM)

This is part of the DOM level 2 user interface event set.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape version – 6.0	
Inherits from:	`Event object`	
JavaScript syntax:	N	*myUIEvent* = new UIEvent()
Object properties:	`detail, view`	
Object methods:	`initUIEvent()`	

The availability of the `UIEvent` object handling can be determined with the `Implementation.hasFeature()` method call.

The available set of events is defined by HTML 4.0 and DOM level 0 with some additional events added. These event types are enumerated in the DOM level 2 specification and are:

- DOMFocusIn
- DOMFocusOut
- DOMActivate

See also:	`AbstractView` object, `Event` object, `Implementation.hasFeature()`, `MouseEvent` object, `onBlur`, `onClick`, `onFocus`

Property	JavaScript	JScript	N	IE	Opera	DOM	Notes
`detail`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly
`view`	1.5 +	-	6.0 +	-	-	2 +	ReadOnly

Method	JavaScript	JScript	N	IE	Opera	DOM	Notes
`initUIEvent()`	1.5 +	-	6.0 +	-	-	2 +	-

Inheritance chain:

`Event` object

UIEvent.detail (Property)

Some detailed information about the `Event` object is made available as a numeric value.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape version – 6.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myUIEvent*`.detail`

The value provided here depends on the `Event` type being inspected.

Property attributes:

`ReadOnly`.

UIEvent.initUIEvent() (Method)

After creating a `UIEvent` object with `document.createEvent()`, it must be initialized with this method call.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape version – 6.0	
JavaScript syntax:	N	*myUIEvent*.initUIEvent(*aType*, *canBubble*, *canCancel*, *aView*, *aDetail*)
Argument list:	*aType*	A string containing the event type
	canBubble	A boolean flag indicating whether the event can bubble
	canCancel	A boolean flag indicating whether the event can be cancelled
	aView	A reference to an `AbstractView` object
	aDetail	A numeric detail value

A new `event` object is manufactured by calling the `DocumentEvent.createEvent()` method. That event should have been defined with a type specified as "`UIEvent`". If it was, then it will support an `initUIEvent()` method. This must be called before the event is dispatched, otherwise the `event` object will not contain enough information for the event dispatcher/handler to make sense of it and route it to the correct target objects.

The two boolean argument values define whether the event will be allowed to be cancelled, and what type of propagation to use (`bubble` or `capture`).

The view argument refers to an `AbstractView` object which DOM level 2 describes and which may not yet be well supported by any browser.

The detail value can be used to pass context information into the event handling chain.

See also:	`DocumentEvent.createEvent()`, `MouseEvent.initMouseEvent()`

UIEvent.view (Property)

A reference to the `AbstractView` object that generated the event.

Availability:	DOM level – 2 JavaScript – 1.5 Netscape version – 6.0	
Property/method value type:	`AbstractView` object	
JavaScript syntax:	N	`myUIEvent.view`

A abstract view object is defined in the DOM level 2 views module. The support for this property in Netscape 6.0 should extend to it providing a `Window` object. The support may not extend to the complete view interface that allows you to select alternate document views.

Property attributes:

`ReadOnly`.

UL object (Object/HTML)

An object that encapsulates an unordered list in a `<UL>` tag.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape version – 6.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	`myUL = myDocument.all.anElementID`
	IE	*myUL* `= myDocument.all.tags("UL")[`*anIndex*`]`
	IE	*myUL* `= myDocument.all[`*aName*`]`
	-	*myUL* `= myDocument.getElementById(`*anElementID*`)`
	-	*myUL* `= myDocument.getElementsByName (`*aName*`)[`*anIndex*`]`
	-	*myUL* `= myDocument.getElementsByTagName ("UL")[`*anIndex*`]`
HTML syntax:	`<UL> ... </UL>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an `Element` object
Object properties:	`compact, start, type`	
Event handlers:	`onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart`	

The `<UL>` tag is a block-level tag. That means that it forces a line break before and after itself.

The DOM level 1 standard describes this as a `UListElement` object.

See also:	DIR object, Element object, OL object

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`compact`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`start`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
`type`	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`onClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDblClick`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onDragStart`	-	3.0 +	-	4.0 +	-	-	-	-
`onFilterChange`	-	3.0 +	-	4.0 +	-	-	-	-
`onHelp`	-	3.0 +	-	4.0 +	-	-	-	Warning
`onKeyDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyPress`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onKeyUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseDown`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseMove`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOut`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseOver`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onMouseUp`	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
`onSelectStart`	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object.

UL.compact (Property)

A switch that compacts the unordered list content to occupy less space on the screen.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape version – 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`myUL.compact`

The collection of `LI` objects are presented in the normal spaced-out style when the compact property belonging to their owner `OL` object is set to `false`.

Setting the property to `true` should result in the list items being squeezed closer together. However, the functionality is rarely supported on web browsers.

It's more likely that you'll apply CSS style attributes to the list to achieve the same effect.

UL.type (Property)

A control for the type of listing style that is presented.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape version – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUL*`.type`

Although an unordered list does not support a sequence numbering scheme, they may be displayed in a variety of different formats selected by this property.

You can override this on an item by item basis and so this property is related to the `LI.type` property.

The following values are appropriate for this property:

- circle
- disc
- square

See also:	`LI.type`, `OL.type`, `style.listStyleType`

Unary expression (Definition)

An operator prefixing an operand.

Availability:	ECMAScript edition – 2

Unary expressions can also be considered to be prefix expressions and also operate on `LValues`.

See also:	Add (+), `delete`, Expression, Logical `NOT` – complement (!), Prefix decrement (--), Prefix increment (++), typeof, Unary operator, `void`

Cross-references:

ECMA 262 edition 2 – section – 11.4

ECMA 262 edition 3 – section – 11.4

Unary operator (Definition)

An operator that requires only one operand to create an expression.

Availability:	ECMAScript edition – 2

Unary operators are those that operate on a single value or expression result.

Some of the unary operators use keywords to convey their meaning. Others are sequences of punctuation characters. Here are the unary operators defined by ECMAScript:

Operator	Meaning
+	Convert the operand to a numeric value.
++	Increment the numeric value of the operand.
-	Convert the operand to a numeric value and negate it.
--	Decrement the numeric value of the operand.
~	Convert the operand to a 32 bit integer and perform a bitwise complement on it.
!	Convert the operand to a boolean value and reverse its value.
new	Invokes an object constructor
delete	Used to delete a property from an object if it can be deleted.
void	Regardless of the result of evaluating the expression that may be operated on, this will always yield the undefined value.

Some of these operators have a different meaning when used with more than one operand.

See also:	Add (+), Bitwise `NOT` – complement (~), delete, Expression, Logical `NOT` – complement (!), Negation operator (-), Operator, Operator Precedence, Positive value (+), Postfix operator, Prefix operator, Ternary operator, typeof, Unary expression, `void`

Cross-references:

ECMA 262 edition 2 – section – 11.4

ECMA 262 edition 3 – section – 11.4

Wrox *Instant JavaScript* – page – 19

undefined (Constant/static)

A literal constant whose type is a built-in primitive value.

Availability:	ECMAScript edition – 2 JavaScript – 1.3 JScript – 5.5 Internet Explorer – 5.5 Netscape version – 4.06
Property/method value type:	`Undefined primitive`

The undefined value is a primitive value that is returned when a variable has not yet been assigned a value.

In some implementations this value compares equal to the `null` value. They are not actually the same, but this can sometimes provide a work-around for those implementations that do not provide a way to explicitly test for an undefined value.

If a variable is declared and no value is assigned to it, then it will contain the undefined value. It will exist and can be referred to in expressions and will affect them to the extent they can be affected by the value 'undefined'.

If a variable has never been declared and it is referred to, a run-time error will result. It does not exist let alone contain the 'undefined' value. There is one instance where referring to a non-existent variable does not generate an error, and that is to use it as an `LValue` in an assignment. Only the = operator can be used. The prefix and postfix increment/decrement operators should yield an error and so should the compound assignment operators, since the value to the left does not exist until after the assignment has taken place.

You cannot create an 'undefined' value explicitly in older interpreters since there is no keyword to yield it as a constant. This is more of a problem with MSIE which only supported the undefined keyword as of version 5.5 while Netscape has supported it since JavaScript 1.3. If you need to, you can manufacture one of your own with this expression:

```
undefined = (void 0);
```

This whole area is somewhat mysterious and the (`void 0`) simulation really creates a `null` value which some browsers cannot distinguish from the undefined value.

Warnings:

- Some implementations do not provide adequate protection against you corrupting this value. Be careful not to assign your own values to this variable. It will lead to unpredictable results if you do.
- This is not available for use server-side with Netscape Enterprise Server 3.

Example code:

```
// Assuming a or b has not been declared, this generates an error
a += 10;
// This doesn't
a = 10;
// This does
alert(b);
```

See also:	Exception, `Global` object, `null`, Null literal, Range error, `void`

Cross-references:

ECMA 262 edition 2 – section – 4.3.9

ECMA 262 edition 3 – section – 4.3.9

O'Reilly *JavaScript Definitive Guide* – page – 47

Undefined behaviour (Definition)

Erroneous operations not described in the standard.

Undefined behavior is what happens when something goes wrong but the standard does not provide any suggested behavior. In many languages, an attempt to divide by zero would yield an undefined result. However JavaScript defines the `NaN` value for just such an occasion.

These situations generally manifest themselves as a run-time error which at the least would present an alert to the user. Sometimes the implementation can catch these errors and may offer the opportunity to stop executing scripts on that web page.

In some cases, the undefined behavior may degenerate to the worst case scenario where the hosting application or even the environment itself is compromized and the system crashes. An example of this is some versions of Netscape when presented with an `<OBJECT>` tag and some versions of MSIE when `onMouseover` events cross from one DOM object to another. In both cases, the browser crashes and does so in such a way that the underlying operating system is also stalled, necessitating a complete reboot.

A portable script should not depend on any undefined behavior performing in a predictable way, even if that behavior is benign. It is very likely that any undefined behavior will not survive between versions of the implementation and almost certainly will not be portable across platforms.

See also:	Behavior, Portability, Range error

undefined type (Type)

A native built-in type.

Availability:	ECMAScript edition – 2
Property/method value type:	`Undefined primitive`

The undefined type has exactly one value, called undefined. It is returned by variables that have not yet been assigned with a value.

This value is also returned in some browsers when referring to a part of the document object model that is non-existent.

Sending a subsequent message to values that are currently undefined results in a run-time error.

See also:	Cast operator, Special type, Type

Cross-references:

ECMA 262 edition 2 – section – 4.3.10

ECMA 262 edition 2 – section – 8.1

ECMA 262 edition 3 – section – 4.3.10

ECMA 262 edition 3 – section – 8.1

Wrox *Instant JavaScript* – page – 14

Undocumented features (Definition)

Aside from the features covered in official documentation, browsers support many other hidden capabilities.

Warnings:

- There have always been undocumented features in the web browsers. Some of these were uncovered during the preparation of this book and have been covered in their own topics.
- Generally, the undocumented features are methods and properties of objects that are not commonly used and maybe an extension to the object that is platform specific. There are also objects that are obscure and can only be discovered by inspection.
- Reading the standards documents, release notes, and manufacturer manuals, only reveals part of the picture.
- The most powerful tool for uncovering these undocumented features is JavaScript itself. In particular the `for( ... in ... )` enumerator which will scan an object for any enumerable properties. This won't reveal all the hidden features but it will uncover many of them.
- Armed with this knowledge, you can write inspection scripts to display the values of those properties. Often they point at an object and if the property is undocumented, then it's likely the object is as well.
- You can spend many hours examining a new browser release and learn much about how it's organized internally. This helps to better understand DOM navigation and how to do 'extreme JavaScript programming'.
- MSIE version 5.0 and 5.5 both introduce many interesting and undocumented features. Netscape version 6.0 builds on that, and adds many more, in particular the sidebar. A great deal of Netscape's user interface is written in JavaScript and by examining the Mozilla source code and the `.js` files that are installed with it, you can learn and possibly customize your browsing experience.

See also:	Netscape

unescape() (Function/global)

Un-URL-escape a string.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0 Deprecated	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`unescape(`*`anInputString`*`)`
Argument list:	*`anInputString`*	A string to be converted

This function is the complement of the `escape()` function described in its own topic elsewhere.

A string that might have been escaped with the `escape()` function either locally or remotely, can be converted back to a normal unescaped string with this function.

This function has Unicode support in MSIE version 4.

As far as ECMAScript is concerned, this is superceded in edition 3 with a set of generalized URI handling functions. The JScript 5.5 documentation refers to this as a deprecated feature.

See also:	Cast operator, `decodeURI()`, `decodeURIComponent()`, `encodeURI()`, `encodeURIComponent()`, `escape()`, `Function` property, `Global` object, URI handling functions

Property attributes:

`DontEnum`.

Cross-references:

ECMA 262 edition 2 – section – 15.1.2.5

ECMA 262 edition 3 – section – B.2.2

Unicode (Standard)

A character encoding standard.

Availability:	ECMAScript edition – 2

The Unicode standard is derived originally from ASCII. ASCII was an 8 bit coded character set with a limited number of individual character values. Unicode is a 16 bit coding that supports all the required characters for the major languages of the world, plus the technical symbols in common use.

Implementations of JavaScript that conform to the ECMA 262 standard must interpret character values in accordance with the Unicode Standard, version 2.0, and ISO/IEC 10646-1 with UCS-2 (Universal Character Set) as the adopted encoding form, implementation level 3. If the adopted 10646 subset is not indicated, then it should be assumed to be the BMP subset, collection 300.

For most usage, the character set will be the lower 128 characters that roughly correspond to the ASCII character table. However, internationalization and localization work in progress suggests that non-English speaking users should be able to declare identifiers in their own natural language with its particular character sets and special symbols. Internet web site domain name standards are undergoing some revision to support double byte characters. The ECMAScript edition 3 improves support for handling localized strings, numbers and dates.

The Unicode standard is due for updating to a new edition as the version 2.0 is somewhat old now and does not support Euro currency symbols among other things. A version 3.0 standard is reported to be on the way.

Unicode characters are more properly referred to as code points.

See also: Character set, Character-case mapping, Control character, Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), `isLower()`, `isUpper()`, Less than (<), Less than or equal to (<=), Locale-specific behavior, Multi-byte character, `NOT` Equal to (!=), `NOT` Identically equal to (!==), String, String literal

Cross-references:

ECMA 262 edition 2 – section – 2

ECMA 262 edition 2 – section – 6

ECMA 262 edition 3 – section – 2

ECMA 262 edition 3 – section – 6

O'Reilly *JavaScript Definitive Guide* – page – 30

Web-references:

ftp://unicode.org/mail_list://unicode-request@unicode.org/
mailto://unicode-inc@unicode.org/
http://www.unicode.org/

Universal coordinated time (Standard)

A universal standard time that is synchronized to GMT.

The universal coordinated time (AKA Universel Temps Cordonne – UTC) is a standard time value that is based on an atomic clock, measured and calibrated according to astronomical observations of pulsars.

UTC is synchronized to Greenwich Mean Time which is coincident with the location of the zero meridian.

Local time at any point on the earth's surface is computed relative to UTC by adding a positive or negative offset to UTC. In general, the offsets are measured in complete hours of 60 minutes duration.

However there are some locations in the world where the offset is a fractional offset in hours. As a general rule, each hour corresponds to a movement of 15 degrees further away from the meridian. This is not completely strict since China spans almost 4 hours in terms of distance but is entirely in one time-zone based on the time in Peking.

Some Islamic countries do not adopt the same time scale in any case, choosing instead to measure the time from sunrise to sunset as 12 hours. Since days wax and wane according to the season, this can cause some difficulty in converting local time to UTC and vice versa.

See also:	Broken down time, Calendar time, Date and time, Daylight savings time adjustment, `java.util.Date`, Time value

UniversalBrowserAccess (Security privilege)

A combination of both read and write access.

This is a combination of everything that `UniversalBrowserRead` and `UniversalBrowserWrite` provide access control for in a single privilege.

See also:	about: URL, `Event` object, `Event.data`, `History` object, `netscape.security.PrivilegeManager`, `onDragDrop`, `PrivilegeManager` object, `PrivilegeManager.disablePrivilege()`, `PrivilegeManager.enablePrivilege()`, Requesting privileges, Restricted access, Same origin, Security policy, self, `Window.close()`

UniversalBrowserRead (Security privilege)

A privilege to grant read access to browser internals.

Gives a script permission to read the properties of windows whose content comes from different origins. This also controls script access to the `history` object among other things. Other internal browser state information that is accessible via the `about:` URL is also controlled by this privilege.

This privilege is also necessary to read the `data` property of an `event` object that is passed to the `ondragdrop()` event handler.

See also:	about: URL, Event object, Event.data, History object, History.current, netscape.security.PrivilegeManager, onDragDrop, PrivilegeManager object, PrivilegeManager.disablePrivilege(), PrivilegeManager.enablePrivilege(), Requesting privileges, Restricted access, Same origin, Security policy

UniversalBrowserWrite (Security privilege)

A privilege that grants the ability to write to internal browser values.

This property allows scripts to close windows. Without this privilege, if a script opens a window, it cannot create a window that is smaller than 100 pixels square. Scripts without this privilege also cannot move a window off-screen, create one that is larger than the screen or create a window that lacks a title-bar. Conferring this privilege allows scripts to do all these things. Without it, scripts cannot hide from view or carry on running when the user thinks they have stopped.

This privilege grants the script permission to hide or show the various window furniture (menu-bar, status-line, scroll-bars, tool-bar, location-bar, directory-bar or personal-bar). Without it, scripts cannot change the visibility of these items.

It also affects event management. You cannot watch events in other windows if they come from different sources without this privilege and you cannot set event object properties without it either. The occasional bug in a browser lets you get round the security in odd ways.

This is how to put a toolbar back onto a window that didn't have one. You are supposed to require privileges to do it, but it sometimes works regardless:

```
open('', '_top', 'status=0,toolbar=1');
```

Putting a toolbar back lets you view source because the menu is reinstated too.

See also:	netscape.security.PrivilegeManager, PrivilegeManager object, PrivilegeManager.disablePrivilege(), PrivilegeManager.enablePrivilege(), Requesting privileges, Restricted access, Same origin, Security policy

UniversalFileRead (Security privilege)

A privilege to grant access to read a file in the file system.

If your script has this privilege, it can then select a file in the client file-system and upload it to the server by means of a FileUpload object. This is done by setting the value property of that FileUpload object.

If this capability is activated, the contents of any file on the local file-system can be presented to a CGI script running on a web server.

See also:	FileUpload object, FileUpload.value, netscape.security.PrivilegeManager, PrivilegeManager object, PrivilegeManager.disablePrivilege(), PrivilegeManager.enablePrivilege(), Requesting privileges, Restricted access, Security policy

UniversalPreferencesRead (Security privilege)

A privilege that allows access to preferences information.

This privilege controls read access to preference settings. Scripts without it cannot see user preference settings. It arbitrates the use of the `Navigator.preference()` method in read mode.

See also:	`History.next`, `History.previous`, `Navigator.preference()`, `netscape.security.PrivilegeManager`, `PrivilegeManager` object, `PrivilegeManager.disablePrivilege()`, `PrivilegeManager.enablePrivilege()`, Requesting privileges, Restricted access, Security policy

UniversalPreferencesWrite (Security privilege)

A privilege that grants the ability to change preferences settings.

This privilege controls whether scripts can change the current user preference settings. It controls the use of the `Navigator.preference()` method.

See also:	`Navigator.preference()`, `Navigator.savePreferences()`, `netscape.security.PrivilegeManager`, `PrivilegeManager` object, `PrivilegeManager.disablePrivilege()`, `PrivilegeManager.enablePrivilege()`, Requesting privileges, Restricted access, Security policy

UniversalSendMail (Security privilege)

A privilege that grants the permission to send mail.

This controls whether a `mailto:` or `news:` URL can be used when submitting a form. A confirmation dialog affirmative response is required if this privilege is not granted to the script.

The result of this is that an e-mail or Usenet news article is submitted that contains the user's e-mail address.

See also:	`mailto: URL`, `netscape.security.PrivilegeManager`, `news: URL`, `PrivilegeManager` object, `PrivilegeManager.disablePrivilege()`, `PrivilegeManager.enablePrivilege()`, Requesting privileges, Restricted access, Security policy

Unspecified behavior (Definition)

Correct behavior that is not specified in the standard.

Although the standard attempts to cover every possible eventuality, there will be parts of an implementation that are not described in the standard. For example, the standard may not define the order in which arguments to a function are to be processed. Some implementations may process them from left to right and others from right to left.

See also:	Behavior

untaint() (Function/global)

A deprecated method for controlling secure access to data values.

Availability:	JavaScript – 1.1 Netscape version – 3.0 Deprecated

This was removed at version 1.2 of JavaScript. If you encounter it in a script you are maintaining, it is probably wise to seek to have it removed, otherwise it is likely to cause a run-time error.

Warnings:

- DO NOT USE THIS FUNCTION!

See also:	`Navigator.taintEnabled()`, `taint()`

unwatch() (Function/global)

Un-set a watch-point for a named property of an object.

Availability:	JavaScript – 1.2 Netscape version – 4.0	
JavaScript syntax:	N	*myObject*`.unwatch(aProperty)`
Argument list:	*aProperty*	A property to cease watching

This method is provided to ease the task of debugging JavaScript.

It is provided to disconnect the watcher that was set up with the `watch()` method.

Its calling circumstances are identical and you simply need to provide a complementary `unwatch()` to deactivate the effects of a `watch()`.

The new event model supported by Netscape version 6.0 and that already available in MSIE 5.0 present a `propertyName` property that belongs to the `Event` object. You can inspect that during an `onPropertyChanged` event and achieve the same `watch()/unwatch()` behavior.

See also:	Event, Event handler, Event management, Event model, `Event` object, `onPropertyChange`, `watch()`

Cross-references:

Wrox *Instant JavaScript* ISBN 1-861001-27-4 – page – 56

URI (Definition)

A Uniform Resource Identifier.

This is another name for a URL. Some commentators argue that a URI is different to a URL. One identifies the target while the other locates it. There may be several copies at different URLs but perhaps they would all be instances of the same URI.

A URI is composed like this (according to the ECMA standard):

```
scheme:first/second;third?fourth
```

See also:	UDI, URL, URN

Cross-references:

RFC 1738, 1808

URI handling functions (Definition)

ECMA edition 3 describes a set of functions it refers to as URI handling properties.

Availability:	ECMAScript edition – 3

According to the ECMA standard (edition 3), a URI is composed of component parts separated by special characters. The special characters are:

- Colon (:)
- Slash (/)
- Semi-colon (;)
- Question-mark (?)

Other characters are reserved for future use.

A URI will need to be encoded without damaging these characters. Normal `escape()` function techniques are not URI safe.

ECMA defines functions for encoding and decoding the entire URI and also functions for dealing with its components individually.

See also:	`decodeURI(),decodeURIComponent(),encodeURI(), encodeURIComponent(),escape(),unescape()`

Cross-references:

ECMA 262 edition 3 – section – 15.1.3

URIError object (Object/core)

A native error object based on the `Error` object.

Availability:	ECMAScript edition – 3 JavaScript – 1.5 Netscape version – 6.0	
JavaScript syntax:	N	*myError* = new URIError()
	N	*myError* = new URIError(*aNumber*)
	N	*myError* = new URIError(*aNumber*, *aText*)
Argument list:	*aNumber*	An error number
	aText	A text describing the error
Object properties:	description, message, name, number	
Object methods:	toString()	

This sub-class of the `Error` object is used when an exception is caused by one of the URI handler functions being used inappropriately.

See also:	`catch( ... )`, `Error` object, `EvalError` object, `RangeError` object, `ReferenceError` object, `SyntaxError` object, throw, try ... catch ... finally, `TypeError` object

Property	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
description	1.5 +	-	6.0 +	-	-	-	3 +	-
message	1.5 +	-	6.0 +	-	-	-	3 +	-
name	1.5 +	-	6.0 +	-	-	-	3 +	-
number	1.5 +	-	6.0 +	-	-	-	3 +	-

Method	JavaScript	JScript	N	IE	Opera	NES	ECMA	Notes
toString()	1.5 +	-	6.0 +	-	-	-	3 +	-

Cross-references:

ECMA 262 edition 3 – section – 15.1.4.15

ECMA 262 edition 3 – section – 15.11.6.6

URL (Definition)

A Uniform Resource Locator.

The URL is composed of several parts. The `http:` in the URL specifies a URL method, which is to be used for retrieving the document or performing the action. There are several alternatives.

The following URL methods may be found. Note that some of these are implementation dependent:

- http: – Hypertext Transfer Protocol. See RFC 2068
- https: – Secure Hypertext Transfer Protocol.
- ftp: – File Transfer Protocol. See RFC 1738
- news: – USENET news. See RFC 1738
- snews: – Secure USENET news.
- mailto: – Electronic mail address. See RFC 2368
- telnet: – Reference to interactive sessions. See RFC 1738
- file: – Host-specific file names. See RFC 1738

These are special Netscape URL request methods that invoke internal sub-systems. Some of these may not be fully functional or may need to be activated by configuration options in the prefs.js file:

- nethelp: – Help manager.
- about: – Internal browser resources.
- mailbox: – Mailbox manager.
- view-source: – Source viewer.
- javascript: – JavaScript interpreter.
- livescript: – JavaScript interpreter.
- mocha: – JavaScript interpreter.
- imap: – internet message access protocol. See RFC 2192
- pop3: – Post Office Protocol v3. See RFC 2384
- ldap: – Local Directory Access Protocol.
- ldaps: – Local Directory Access Protocol (secure).
- rlogin: – Remote login.
- tn3270: – Interactive 3270 emulation sessions.
- wais: – Wide Area Information Servers. See RFC 1738
- wysiwyg: – JavaScript generated page content.

The MSIE browser also uses the `clsid:` URL to locate registered ActiveX controls that are loaded with the `<OBJECT>` tag.

The IANA documention also lists these protocols:

- gopher: – The Gopher Protocol. See RFC 1738
- nntp: – USENET news using NNTP access. See RFC 1738
- prospero: – Prospero Directory Service. See RFC 1738

- z39.50s: – Z39.50 Session. See RFC 2056
- z39.50r: – Z39.50 Retrieval. See RFC 2056
- cid: – content identifier. See RFC 2392
- mid: – message identifier. See RFC 2392
- vemmi: – versatile multimedia interface. See RFC 2122
- service: – service location. See RFC 2609
- nfs: – network file system protocol. See RFC 2224
- acap: – application configuration access protocol. See RFC 2244
- rtsp: – real time streaming protocol. See RFC 2326
- tip: – Transaction Internet Protocol. See RFC 2371
- data: – data. See RFC 2397
- dav: – dav. See RFC 2518
- opaquelocktoken: – opaquelocktoken. See RFC 2518
- sip: – session initiation protocol. See RFC 2543
- tel: – telephone. See RFC 2806
- fax: – fax. See RFC 2806
- modem: – modem. See RFC 2806
- afs: – Andrew File System global file names.
- mailserver: – Access to data available from mail servers

These are specified as part of the ATVEF standard to provide support for TV set top boxes:

- tv: – Display a TV picture where you might use an image.
- lid: – Local identifier for accessing cached and carousel delivered assets.
- uhttp: – Unidirectional (broadcast) HTTP content.

Warnings:

- As of autumn 2000, there are changes afoot to allow characters other than the English letters and numbers to be used in URL values. The Verisign organisation will act as an arbiter of whether a URL is valid but it may be composed of arabic and Chinese characters for example.

See also:	about: URL, ATVEF, clsid: URL, file: URL, ftp: URL, http: URL, https: URL, javascript: URL, mailbox: URL, mailto: URL, nethelp: URL, news: URL, snews: URL, telnet: URL, UDI, URI, URN, view-source: URL

Url object (Object/HTML)

An object that represents URLs in Netscape but which has extensions in MSIE.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Inherits from:	`Element object`	
JavaScript syntax:	-	*myUrl* = *myDocument*.links[*anIndex*]
Argument list:	*anIndex*	A reference to an element in a collection
Object properties:	`charset, coords, hash, host, hostname, href, hreflang, Methods, mimeType, name, nameProp, pathname, port, protocol, protocolLong, rel, rev, search, shape, tabIndex, target, text, type, urn, x, y`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

Objects of this type are contained in the `document.links[]` array.

Event handling support via properties containing function objects was added to `Url` objects at version 1.1 of JavaScript.

See also:	Anchor object, Area object, Document.links[], HyperLink object, LINK object, LinkArray object, Location object, String.anchor(), String.link(), Url.hash, Url.host, Url.hostname, Url.href, Url.name, Url.pathname, Url.port, Url.protocol, Url.search, Url.target, Url.text

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`charset`	-	-	-	-	-	-	-	-
`coords`	1.0 +	1.0 +	2.0 +	3.02 +	-	-	-	-
`hash`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
`host`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	-
`hostname`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
`href`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
`hreflang`	-	-	-	-	-	-	-	-
`Methods`	-	3.0 +	-	4.0 +	-	-	-	-
`mimeType`	-	3.0 +	-	4.0 +	-	-	-	-
`name`	1.5 +	1.0 +	6.0 +	3.02 +	-	-	-	Warning
`nameProp`	-	3.0 +	-	4.0 +	-	-	-	-
`pathname`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
port	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
protocol	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	-
protocolLong	-	3.0 +	-	4.0 +	-	-	-	-
rel	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
rev	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
search	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	-
shape	1.0 +	1.0 +	2.0 +	3.02 +	-	1 +	-	-
tabIndex	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	-
target	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	-
text	1.2 +	-	4.0 +	-	-	-	-	Warning
type	1.5 +	3.0 +	6.0 +	4.0 +	-	1 +	-	-
urn	-	3.0 +	-	4.0 +	-	-	-	-
x	1.2 +	-	4.0 +	-	-	-	-	Warning, Deprecated
y	1.2 +	-	4.0 +	-	-	-	-	Warning, Deprecated

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onDblClick	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyPress	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onKeyUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseDown	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	4.0 +	Warning
onMouseOver	1.0 +	1.0 +	2.0 +	3.0 +	3.0 +	-	4.0 +	Warning
onMouseUp	1.2 +	3.0 +	4.0 +	4.0 +	3.0 +	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

Url.charset (Property)

The character set of the document pointed at by the `HREF="…"` HTML tag attribute of an anchor.

Property/method value type:	String primitive	
JavaScript syntax:	none	*myUrl*.charset

This would contain the character set being used by the document referred to by the `HREF="..."` HTML tag attribute. For example the value "`iso-8859-1`" is likely to be returned but the local variant of the browser and OS may affect the value you get.

This property might contain a value such as:

```
csISO5427Cyrillic
```

Details of other aliases can be located at the IANA registry.

See also:	Anchor.charset, LINK.charset

Url.coords (Property)

The co-ordinates on screen where the object represented by the URL is located.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myUrl*.coords

When a shaped area is defined within an image map, the extent rectangle around the shape is defined with the co-ordinates property. The value is defined with the `COORDS="..."` HTML tag attribute.

See also:	Anchor.coords, Url.shape

Url.hash (Property)

The hash delimited item at the end of an HREF URL if there is one.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*.hash
	-	*myUrl*.hash = *aLocation*
HTML syntax:	`<A HREF="..."><LINK HREF="...">`	
Argument list:	*aLocation*	A new named location within the document

This property yields a text string that contains the hash suffix from the URL if there is one.

You can assign a new value to this property, which will become a new anchor location within the document.

Warnings:

- This attribute may not work correctly when URLs are accessed from one frame to another in some versions of MSIE. You should check your target platforms for compliance.
- If you assign a value to this property, you should omit the leading hash.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE" HREF="http://www.mydomain.com/folder/file.html#abcdef">Click
here</A><BR>
<SCRIPT>
//In testing, this did not work on Opera 5.0
document.write(document.links[0].hash);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor.hash`, URL, `Url` object, `Url.href`, `Url.pathname`, `Url.port`, `Url.protocol`, `Url.search`, `Url.target`

Url.host (Property)

The hostname and port number of an HREF URL if there is one.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*.host
	-	*myUrl*.host = aHost
HTML syntax:	`<A HREF="..."><LINK HREF="...">`	
Argument list:	*aHost*	A new hostname and optional port value

This property yields the hostname and colon delimited port number of the HREF value in an `<A>` tag if there is one and an empty string if not.

You can redefine the host by assigning a new value to this property.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE"
HREF="http://www.mydomain.com:8080/folder/file.html#abcdef">Click here</A><BR>
<SCRIPT>
document.write(document.links[0].host);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor.hash`, `Anchor.host`, URL, `Url` object, `Url.href`, `Url.pathname`, `Url.port`, `Url.protocol`, `Url.search`, `Url.target`

Url.hostname (Property)

The hostname of an HREF URL if there is one.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myUrl*.hostname
	-	*myUrl*.hostname = *aHostname*
HTML syntax:	<A HREF="..."><LINK HREF="...">	
Argument list:	*aHostname*	The name of a host

This property yields the host value of the HREF value in an <A> tag if there is one and an empty string if not.

You can redefine the hostname by assigning a new value to this property.

Warnings:

- Be careful not to assign a port number with the host name, otherwise your new URL may acquire two port numbers which makes it invalid.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE"
HREF="http://www.mydomain.com:8080/folder/file.html#abcdef">Click here</A><BR>
<SCRIPT>
// In testing, this did not work on Opera 5.0
document.write(document.links[0].hostname);
</SCRIPT>
</BODY>
</HTML>
```

See also:	Anchor.hash, Anchor.hostname, URL, Url object, Url.href, Url.pathname, Url.port, Url.protocol, Url.search, Url.target

Url.href (Property)

The complete HREF of an HREF URL if there is one.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0
Property/method value type:	String primitive

JavaScript syntax:	-	*myUrl*.href
	-	*myUrl*.href = *aHREF*
HTML syntax:	<A HREF="..."><LINK HREF="...">	
Argument list:	*aHREF*	A valid HREF value

This property yields a text string that contains the complete HREF value from the URL if there is one and an empty string if not.

You can redefine the entire HREF content by assigning a new value to this property.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE"
HREF="http://www.mydomain.com:8080/folder/file.html#abcdef">Click here</A><BR>
<SCRIPT>
document.write(document.links[0].href);
</SCRIPT>
</BODY>
</HTML>
```

See also:	Anchor.hash, Anchor.href, Location.href, URL, Url object, Url.hash, Url.host, Url.hostname, Url.pathname, Url.port, Url.protocol, Url.search, Url.target

Url.hreflang (Property)

The language that the document at the HREF location is coded in.

Property/method value type:	String primitive	
JavaScript syntax:	none	*myUrl*.hreflang

This property should contain values that use the international language two-letter abbreviation codes. These are not the same as the country codes, which are also two letter values.

Refer to the Language codes topic for a list of the available codes.

See also:	Language codes, LINK.hreflang

Url.Methods (Property)

A collection of methods belonging to the URL.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Collection object`	
JavaScript syntax:	IE	*myUrl*`.Methods`
HTML syntax:	`<A HREF="..."><LINK HREF="...">`	

The values here would be one of the valid methods for the HTTP protocol. It would be one of the following:

- GET
- HEAD
- POST
- PUT
- DELETE
- OPTIONS
- TRACE

It is likely that only the `GET` and `POST` methods make any logical sense in this contents and on rare occasions, the `PUT` method may be referred to although it is unusual to find a web server that accepts documents with this request method.

See also:	`Anchor.Methods`

Url.mimeType (Property)

The MIME type used to access the document pointed at by the URL.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myUrl*`.mimeType`

The MSIE browser maps the file extension of the file belonging to the anchor to an extended description of the file format, which it makes available through the `mimeType` property. Here is a list of some `mimeType` values it pays special attention to.

File type	MSIE expanded Mime type
.css	Microsoft CSS1 Style Sheet (W3C would have been more appropriate)
.gif	GIF Image
.htc	Microsoft HTML Component file for behaviours
.htm	Microsoft HTML Document 4.0
.html	Microsoft HTML Document 4.0
.jpg	JPEG Image
.js	Microsoft JScript File (this is a bit presumptuous)
.txt	Text Document
.vbs	Microsoft VBScript File
.xxx	All unrecognized file types are returned as xxx File with no further expansion

Microsoft assert that `.htm` and `.html` files are "Microsoft HTML" and `.css` files are "Microsoft CSS1" style sheets. They also assert that `.js` files are "Microsoft JScript" files. This is not strictly true because they don't own those file extensions across all platforms, nor indeed do they even own them on the Windows platform.

See also:	`Anchor.mimeType`

Url.name (Property)

In MSIE, the `Url.name` property is another way to access the `anchor.name` property.

Availability:	JavaScript – 1.5 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*`.name`
HTML syntax:	`<A NAME="aName">`	
Argument list:	*aName*	A name for an anchor

The value of this property is defined by the `NAME="..."` tag attribute in the HTML that describes the document. Without the `NAME` attribute the anchor object does not get added to the `anchors[]` array.

Warnings:

- This only works on MSIE and not in Netscape. This is because Netscape maintains links and anchors as strictly different things altogether. MSIE treats them as the same class of object.
- Logically there is not much purpose in changing the name of a link anyway.
- Be aware in MSIE that assigning a new name will affect the length of the `document.anchors[]` array even though the item is rightly a `link` object.

See also:	`Anchor.name`, `Area.name`, `Document.links[]`, `LinkArray.length`, `Url` object

Url.nameProp (Property)

The filename portion of the URL value.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myUrl.nameProp`

This property extracts the filename portion of the `HREF="..."` value for this anchor object.

See also:	`Anchor.nameProp`

Url.pathname (Property)

The pathname portion of an HREF URL if there is one.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*`.pathname`
	-	*myUrl*`.pathname = aPath`
HTML syntax:	`<A HREF="..."><LINK HREF="...">`	
Argument list:	*aPath*	A new pathname value

This property yields a text string that contains the pathname value from the URL if there is one and an empty string if not.

MSIE and Netscape support the use of this property as an `LValue`. If you write to it, the pathname portion of the HREF value is modified. Be careful not to include a hash or search/query value.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE"
HREF="http://www.mydomain.com:8080/folder/file.html#abcdef">Click here</A><BR>
<SCRIPT>
document.write(document.links[0].pathname);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor.hash`, *`Anchor.pathname`*, URL, `Url` object, `Url.hash`, `Url.host`, `Url.hostname`, `Url.href`, `Url.port`, `Url.protocol`, `Url.search`, `Url.target`

Url.port (Property)

The port value of an HREF URL if there is one.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myUrl`*`.port`
	-	*`myUrl`*`.port = `*`aPortNumber`
HTML syntax:	`<A HREF="..."><LINK HREF="...">`	
Argument list:	*`aPortNumber`*	A numeric value to be used as a port number

This property yields the port number value of the HREF attribute in an `<A>` tag if there is one and an empty string if not.

You can assign a value to this property as if it was an `LValue`.

Warnings:

- Do not include the delimiting colon when you assign a value to this property.
- Make sure you assign a numeric value. Non-numeric values will be rejected. This is to avoid the possibility of a completely invalid port number being used.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE"
HREF="http://www.mydomain.com:8080/folder/file.html#abcdef">Click here</A><BR>
<SCRIPT>
// In testing, this did not work in Opera 5.0
document.write(document.links[0].port);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor.hash`, `Anchor.port`, URL, `Url` object, `Url.hash`, `Url.host`, `Url.hostname`, `Url.href`, `Url.pathname`, `Url.protocol`, `Url.search`, `Url.target`

Url.protocol (Property)

The protocol value of an HREF URL if there is one.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*`.protocol`
	-	*myUrl*`.protocol = aProtocol`
HTML syntax:	`<A HREF="..."><LINK HREF="...">`	
Argument list:	*aProtocol*	A valid protocol for the URL that the browser supports

This property yields text string that contains the protocol value from the URL if there is one and an empty string if not.

Using this property as an `LValue`, you can redefine the protocol for the link if it has an HREF. You might want to do this if you want to change the way you access a particular document.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
```

```
<A NAME="EXAMPLE"
HREF="http://www.mydomain.com:8080/folder/file.html#abcdef">Click here</A><BR>
<SCRIPT>
// In testing, this did not work in Opera 5.0
document.write(document.links[0].protocol);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor.hash`, `Anchor.protocol`, `IMG.protocol`, URL, `Url` object, `Url.hash`, `Url.host`, `Url.hostname`, Url.href, `Url.pathname`, `Url.port`, `Url.search`, `Url.target`

Url.protocolLong (Property)

A long, human readable description of the protocol.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*`myUrl`*`.protocolLong`

Only the MSIE browser supports this, and it's of such limited use as to make one question its purpose.

See also:	`Anchor.protocolLong`

Url.rel (Property)

The relationship of this element to the document pointed at by the URL.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape version – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myUrl.rel`
HTML syntax:	`<LINK REL="...">`	

This is sometimes called a forward link. Although the `HREF="..."` HTML tag attribute is normally the only means used to identify a target document, the browser is permitted to use the `REL="..."` HTML tag attribute to decide whether to use the HREF value or how it should be used.

The following HTML version 4.0 standard link types are permitted in this property:

- alternate
- appendix
- bookmark
- chapter
- contents
- copyright
- glossary
- help
- index
- next
- prev
- section
- start
- stylesheet
- subsection

MSIE adds these as well:

- same
- next
- parent
- previous

When used or tested within a script, any comparisons should be case insensitive.

See also:	`LINK.rel`

Url.rev (Property)

The relationship of the document pointed at by the URL to this element.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape version – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*`.rev`

This is sometimes called a reverse link. It defines the relationship between a document and another that calls it. The linkage is defined from the destination document's viewpoint.

This property supports the same HTML version 4.0 standard link types as the `rel` property. Refer to that topic for details.

When used or tested within a script, any comparisons should be case insensitive.

Because `rel` and `rev` properties are complementary, the values in them are likely to be related. For example, if one contains the value "`next`" then the other is likely to contain "`previous`".

See also:	`LINK.rev`

Url.search (Property)

The query portion of an HREF URL if there is one.

Availability:	JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*`.search`
	-	*myUrl*`.search = `*aString*
HTML syntax:	`<A HREF="..."><LINK HREF="...">`	
Argument list:	*aString*	A new search string.

This property yields a text string that contains the query value from the URL if there is one and an empty string if not.

This should really be called the query portion of the URL instead of the search attribute.

You can assign a new value to this property. This might be useful if the user needs to specify some search parameters, which can be range checked at the client end. You will need to prefix the new value with a question mark otherwise the assignment won't work.

It is probably named 'search' due to it having been used by search engines, and that being the way people first noticed its widespread use when inspecting their web server's referrer logs.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE"
HREF="http://www.mydomain.com:8080/folder/file.html#abcdef">Click here</A><BR>
<SCRIPT>
// In testing, this did not work in Opera 5.0
document.write(document.links[0].search);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor.hash`, `Anchor.search`, `request.<urlExtension>`, URL, `Url` object, `Url.hash`, `Url.host`, `Url.hostname`, `Url.href`, `Url.pathname`, `Url.port`, `Url.protocol`, `Url.target`

Url.shape (Property)

The shape of an area in an image map that this URL is associated with.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`myUrl.shape`

This property has a meaningful value when the Url object is instantiated via `<MAP>` and `<AREA>` tag. It defines the shape of the hotspot within the extent rectangle, defined by the `coords` property. It might contain one of the following values:

- default
- rect
- circle
- poly

See also:	`Anchor.shape`, `Area.shape`, `Url.coords`

Url.target (Property)

The target attribute of an <A> tag if there is one.

Availability:	DOM level – 1 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape version – 2.0 Opera browser – 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*.target
	-	*myUrl*.target = *aTarget*
HTML syntax:	<A TARGET="..."><LINK TARGET="...">	
Argument list:	*aTarget*	The name of a new target window or frame

This property yields the value of the TARGET attribute in an <A>, <AREA> or <MAP> tag if there is one and an empty string if not.

You can assign a new value to this property so that the URL will be directed to a different window or frame.

Here are some example target values:

- _parent
- _self
- _top
- _blank
- Window name
- Frame name

Netscape version 6.0 adds the _content target value for use within HTML that exists in the sidebar area. This ensures that the link is targeted at the main window and does not overwrite the sidebar.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A NAME="EXAMPLE" HREF="http://www.mydomain.com:8080/folder/file.html#abcdef"
TARGET="_top">Click here</A><BR>
<SCRIPT>
document.write(document.links[0].target);
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Anchor.hash`, `Anchor.target`, `BASE.target`, `Location.target`, `Map.target`, URL, `Url` object, `Url.hash`, `Url.host`, `Url.hostname`, `Url.href`, `Url.pathname`, `Url.port`, `Url.protocol`, `Url.search`

Url.text (Property)

The text that appears between the `<A>` and `</A>` tags.

Availability:	JavaScript – 1.2 Netscape version – 4.0	
Property/method value type:	`String primitive on Netscape` `Undefined value on MSIE`	
JavaScript syntax:	N	*myUrl*`.text`

This is equivalent to the `innerText` property that MSIE supports. On Netscape 4 `Url.text` is ReadOnly while the `Url.innerText` property on MSIE 4 allows `read` and `write` access.

Assigning to this property in MSIE simply creates a `text` property but does not affect the text of the anchor.

The value yielded by this property (when it does work) is the text between the `<A>` and `</A>` tags.

Warnings:

- You will need to detect the browser type before attempting to use this property.
- Does not work properly on Netscape on the Macintosh.
- Even if it does work, you may only extract a portion of the text from the anchor.

See also:	`Anchor.text`, `Url` object

Url.type (Property)

The MIME type of the document that the URL points at.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 3.0 Internet Explorer – 4.0 Netscape version – 6.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myUrl*`.type`

The MIME type of the document associated with the Url is accessible through the value of this property.

Refer to the MIME type topic for details of the available MIME types you are likely to see in this property.

See also:	`LINK.type`

Url.urn (Property)

A URN value derived from the URL.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	*myUrl*`.urn`

Refer to:

URN

Url.x (Property)

The X co-ordinate of a link.

Availability:	JavaScript – 1.2 Netscape version – 4.0 Deprecated	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myUrl*`.x`

The horizontal position of the object in the display measured in pixels. You can use the x and y co-ordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Warnings:

- This is withdrawn from Netscape 6.0 and no longer supported.

See also:	`Location.x`

Url.y (Property)

The Y co-ordinate of a link.

Availability:	JavaScript – 1.2 Netscape version – 4.0 Deprecated

Property/method value type:	Number primitive	
JavaScript syntax:	N	*myUrl*.y

The vertical position of the object in the display measured in pixels. You can use the x and y co-ordinates of the object as targets of the `scrollTo()` method for the window it lives in.

Warnings:

- This is withdrawn from Netscape 6.0 and no longer supported.

See also:	Location.y

URN (Definition)

A Uniform Resource Name.

The URN notation is an alternative means of formatting the URI portion of a URL. This is a standard which is in the process of being defined and deployed. Details are available in RFC 2141.

The format is somewhat different to that of a URL placed in an `HREF="..."` HTML tag attribute. It is not at present widely supported in browsers but may be in the future.

See also:	UDI, URI, URL

User-generated object (Definition)

An object created under script control.

JavaScript allows the user (or script developer really) to create their own objects, properties and methods. The prototypes for existing built-in objects can also be extended by the script developer.

This has generally been supported since the earliest versions of the main JavaScript interpreters. It was made available in the WebTV platform effective from the Summer 2000 release. This means that DHTML effects created by authoring tools such as DreamWeaver should work better in WebTV boxes that are shipped after this date or are upgraded in the field.

See also:	JellyScript

userDefined object (Object/DOM)

You can define your own objects in several different ways.

Availability:	DOM level – 1 JavaScript – 1.5 JScript – 5.0 Internet Explorer – 5.0 Netscape version – 6.0	
JavaScript syntax:	-	`myObject = new Function()`
	-	`myObject = new Object()`

Aside from the ways in which you can create custom objects with your own constructor functions, you can also create your own HTML tags to make custom document elements.

Objects are instantiated from HTML tags and assume a class name from the HTML tag names in the document. So you can invent your own tags that become objects in the document model in MSIE, and inherit some basic capabilities from the environment. Netscape version introduces this functionality at version 6.0.

See also:	`Object` object

userProfile object (Object/JScript)

An object that encapsulates a user's profile within the system.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`myUserProfile = navigator.userProfile`
Object methods:	`addReadRequest(), clearRequest(), doReadRequest(), getAttribute()`	

Through this object, a script can access details about the user from the profile that the system maintains.

This is accomplished by the arcane method of queueing requests for access. Then having queued all the requests, you can ask the user for permission with a single dialog.

If the user grants permission, you can then access the userProfile values, after which the request queue can be cleared.

Warnings:

- This is a not fully supported on some versions of MSIE for Macintosh.

See also:	`userProfile.addReadRequest(), userProfile.clearRequest(),userProfile.doReadRequest(), userProfile.getAttribute()`

Method	JavaScript	JScript	N	IE	Opera	Notes
`addReadRequest()`	-	3.0 +	-	4.0 +	-	-
`clearRequest()`	-	3.0 +	-	4.0 +	-	-
`doReadRequest()`	-	3.0 +	-	4.0 +	-	-
`getAttribute()`	-	3.0 +	-	4.0 +	-	?

userProfile.addReadRequest() (Method)

Adds a request to read a property of the user profile. These are queued up to have permission requested from the user.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`navigator.userProfile.addReadRequest (anAttribute)`
Argument list:	*`anAttribute`*	One of the valid vCard attribute names

This method adds a request to the queue. Each request denotes the parameters within the user profile that the script needs to access. You may call this method several times, each time asking for a different user preference. You won't initially be granted access to them until you have executed the `doReadRequest()` method. If the user affirms that you can access these values, then on its return, you can use the `getAttribute()` method to access the value you need. You can only access values you have requested and which the user has affirmed.

Here is a list of all the user preference attributes that you can request:

- vCard.Business.City
- vCard.Business.Country
- vCard.Business.Fax
- vCard.Business.Phone
- vCard.Business.State
- vCard.Business.StreetAddress
- vCard.Business.URL
- vCard.Business.Zipcode
- vCard.Cellular
- vCard.Company
- vCard.Department
- vCard.DisplayName
- vCard.Email
- vCard.FirstName
- vCard.Home.City
- vCard.Home.Country
- vCard.Home.Fax
- vCard.Home.Phone
- vCard.Home.State

- vCard.Home.StreetAddress
- vCard.Home.Zipcode
- vCard.Homepage
- vCard.JobTitle
- vCard.LastName
- vCard.MiddleName
- vCard.Notes
- vCard.Office
- vCard.Pager

See also:	`userProfile` object, `userProfile.clearRequest()`, `userProfile.doReadRequest()`,`userProfile.getAttribute()`, `vCard` object

userProfile.clearRequest() (Method)

Clear all read requests from the queue.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`navigator.userProfile.clearRequest()`

You should use this as soon as you have retrieved the necessary user information. It purges all pending requests. If you don't use this call, then if you made some more requests, next time you call the `doReadRequest()` method the user is asked for permission to access an ever increasing list of preferences.

See also:	`userProfile` object, `userProfile.addReadRequest()`

userProfile.doReadRequest() (Method)

Execute the current queue of read requests, asking the user for permission to access these properties of the profile.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`navigator.userProfile.doReadRequest(aCode)`
	IE	`navigator.userProfile.doReadRequest(aCode, aName)`
	IE	`navigator.userProfile.doReadRequest(aCode, aName, aDomain)`
	IE	`navigator.userProfile.doReadRequest(aCode, aName, aDomain, aPath)`
	IE	`navigator.userProfile.doReadRequest(aCode, aName, aDomain, aPath, anExpire)`

Argument list:		
	aCode	The use code selects an explanatory message
	aDomain	A server domain
	aName	A name denoting who is asking
	anExpire	An expiry date for the permissions
	aPath	A document path

Having queued some requests to read user preferences of course you would only request access to ones you genuinely needed. Then you should call this method. It presents the user with a dialog asking for permission to access the listed preference values. If the user decides to grant access, you will have access to the user preferences, but only to the ones you have requested.

There are a set of optional parameters for this method although the first argument is mandatory. These arguments are:

- A usage code. This is a numeric value but it is translated into an explanatory text in the dialog presented to the user. This argument must be present.
- The name argument is available so that you can say who you are, when you ask for the permission to access the user profile.
- You can specify a server domain, so that the permission granted can be recorded for that domain, to save requests from that domain interrupting the user to ask for the same permission more than once. This lasts for the session or until the expiry date if it is set.
- You can specify a document path within which permission granted will apply. This is much like the way that cookies are allowed to be returned only for documents within a path tree. This lasts for the session or until the expiry date if it is set.
- The last optional argument is an expiry date up to which time the permissions will be cached and can be used again subject to the domain and path criteria being satisfied.

Here is a list of the usage codes for use with the first argument:

Code	Description
0	System administrator request.
1	Research and development.
2	Complete current transaction.
3	Modify the site content.
4	Improve site content.
5	Notify about updates to the site.
6	Contacting for sales leads.
7	Linking to other information already collected.
8	Other unspecified purposes.
9	Disclosure to other parties for reasons of updating the site.
10	Disclosure to other parties for marketing purposes.
11	Disclosure to other parties for marketing purposes with your permission.
12	Disclosure to other parties for any other purposes.

See also:	`userProfile` object, `userProfile.addReadRequest()`

userProfile.getAttribute() (Method)

Get an attribute of the `vCard` object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`navigator.userProfile.getAttribute (attribName)`
Argument list:	*`attribName`*	The name of the required VCard attribute

If the user grants access to the `userProfile` data, this method will return the contents of the named attribute. The same attribute names that were used when the request was queued should be used here. These are all member attributes belonging to the `vCard` object.

You should call this method once for each attribute you need. If you have not been granted access to any attribute you will simply receive a `null` value.

Warnings:

- Some documentary sources suggest that this method is largely dysfunctional on a variety of platforms.

See also:	`userProfile` object, `userProfile.addReadRequest()`, `vCard` object

UTC (Standard)

Universel Temps Cordonne.

See also:	Broken down time, Calendar time, Date and time, Daylight savings time adjustment, Universal coordinated time

Utility objects (Definition)

Netscape Enterprise Server provides server-side utility objects to give assistance when running server-side JavaScript.

Refer to:

Netscape Enterprise Server

Value of an expression (Definition)

The result obtained by evaluating an expression.

See also:	LValue, RValue

Value preserving (Definition)

The act of converting values from one data type to another should preserve the value with no loss of resolution.

See also:	Integer promotion

Value property (Definition)

A property value of an object.

Availability:	ECMAScript edition –2
Property/method value type:	`Depends on value type`

A value property returns a generally constant value. Examples are the NaN and Infinity values returned by core objects.

Warnings:

- Do not create your own object properties with the same name as internal or core value properties. If you do, unpredictable behavior may result. Internal values should be protected against accidental damage in this way and they would be, if the interpreter designers were infallible. Unfortunately, to date, they have proven to everyone's dissatisfaction that infallibility is a lost art.

See also:	`Global` object, Infinity, Internal Property, `NaN`

Cross-references:

ECMA 262 edition 2 – section – 15.1.1

ECMA 262 edition 3 – section – 15.1.1

valueOf() (Method)

Returns the primitive equivalent value of the receiving object.

Availability:	JavaScript – 1.1 JScript – 3.0 Internet Explorer- 4.0 Netscape version – 3.0	
Property/method value type:	`Primitive value`	
JavaScript syntax:	-	*myObject*`.valueOf()`

The primitive value of the receiving object is returned by this method.

This method is supported by virtually all objects due to the fact that it is available as a method of the `Global` object in Netscape. Therefore it is inherited into the scope chain for every script and function (method).

Host implementations will generally override this inherited support and provide specialized output routines that yield a reasonable value.

Most likely the value will be a String primitive but it need not be. It could be any of the built-in core primitive data types as specified in the ECMA-262 standard.

See also:	`Array.valueOf()`, `Boolean.valueOf()`, `Date.valueOf()`, Number, `Number.valueOf()`, `Object.valueOf()`, `String.valueOf()`

var (Declaration)

A variable declarator.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer- 3.02 Netscape version – 2.0 Netscape Enterprise Server version – 2.0 Opera browser – 3.0	
JavaScript syntax:	-	`var anIdentifier`
	-	`var anIdentifier = anInitialVaue`
	-	`var anIdentifier = anInitialVaue, anIdentifier = anInitialVaue, ...`
	-	`var anIdentifier, anIdentifier, ...`
Argument list:	*anIdentifier*	The name of a variable.
	anInitialVaue	An initial value for the variable

The `var` keyword prefaces a list of variable declarations. The list is terminated by a semi-colon.

Variables can be initialized in the declaration list that can declare a single variable or several at once.

If the `var` statement occurs inside a function declaration, the variables are defined with a scope that is local to that function body. Otherwise they are defined with global scope and are created as members of the `Global` object. When they are created as global variables, they take on the `DontDelete` property attributes.

Variables are created when an execution scope is entered. A block does not indicate a new execution scope, so variables cannot be created local to a code block unless it is a function body. That means they are not local to an `if(),while()` or `for()` block of compound statements.

When variables are created, they are initialized to contain the value undefined. If an initializer is added to the variable declaration, the value is assigned when the `var` statement is executed, which may be some time after the execution scope has caused the variable to be created.

Warnings:

- If you fail to declare a variable with the `var` keyword, you will end up creating a global variable automatically. If the variable should only have local scope and duration within a function, then you may want to avoid this.
- You should use local variables as often as possible to avoid unwanted side effects.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
var myGlobalVariable = 100;
function myLocalScope()
{
   var myLocalVariable = 200;
}

alert(myGlobalVariable);
alert(myLocalVariable);
</SCRIPT>
</BODY>
</HTML>
```

See also:	= (Assign), Assign value (=), Assignment expression, Assignment operator, Comma operator (,), Compound statement, function(...) ..., `Global` object, Initialisation, Semi-colon (;), Statement, Variable Declaration, Variable statement

Cross-references:

ECMA 262 edition 2 – section – 10.1.3

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 3 – section – 10.1.3

ECMA 262 edition 3 – section – 12.2

VAR object (Object/HTML)

An object representing the HTML content delimited by the `<VAR>` tags.

Availability:	JavaScript – 1.5 JScript – 3.0 Internet Explorer- 4.0 Netscape version – 6.0	
Inherits from:	`Element object`	
JavaScript syntax:	IE	*myVAR* = *myDocument*.all.*anElementID*
	IE	*myVAR* = *myDocument*.all.tags("VAR")[*anIndex*]
	IE	*myVAR* = *myDocument*.all[*aName]*
	-	*myVAR* = *myDocument*.getElementById(*anElementID*)
	-	*myVAR* = *myDocument*.getElementsByName(aName)[*anIndex*]
	-	*myVAR* = *myDocument*.getElementsByTagName("VAR")[*anIndex*]

HTML syntax:	<VAR> ... </VAR>	
Argument list:	*anElementID*	The ID value of the element required
	anIndex	A reference to an element in a collection
	aName	An associative array reference
Event handlers:	onClick, onDblClick, onDragStart, onFilterChange, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onSelectStart	

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDblClick	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onDragStart	-	3.0 +	-	4.0 +	-	-	-	-
onFilterChange	-	3.0 +	-	4.0 +	-	-	-	-
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onKeyDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyPress	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onKeyUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseDown	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseMove	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOut	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseOver	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onMouseUp	1.5 +	3.0 +	6.0 +	4.0 +	-	-	4.0 +	Warning
onSelectStart	-	3.0 +	-	4.0 +	-	-	-	-

Inheritance chain:

`Element` object, `Node` object

See also:	`Element` object

Variable (Definition)

A named storage receptacle for script accessible values.

Availability:	ECMAScript edition – 2

Variables are containers with names that you can manage from your JavaScript code. You can create new ones when you need to, and change the values that are stored in them whenever you want.

Some variables are available only within the function block that is currently executing while others are available all the time. The first kind are called local variables and the second are called global variables.

The availability of a variable is determined by the scope chain rules that are in force at any particular time.

When a primitive value is stored in the variable, a unique copy of that value is associated with the variable. However, when an object is stored in a variable, the variable contains only a reference to the object. The object's reference count is incremented and is only decremented when a reference to it is discarded. To discard the reference to an object that is contained in a variable, it must be deleted explicitly. This will not affect the object but will reduce its reference count by one. When its reference count is zero, the object can be purged from memory. Whether that happens or not depends on the way that garbage collection is managed. It is possible that simply replacing an object reference with some other value may not properly decrease the reference count in some implementations. This can lead to a memory leak.

Example code:

```
// Simple variable declaration.
var myVariable;

// Declare and initialise a variable to a null value.
var emptyVar = null;

// Declare and initialise to a floating point value.
var priceData = 11.50;

// Assign a string value to a variable.
someString = "23 The Irons, Cleethorpes";

// Assign a boolean value to a variable.
aBooleanValue = true;

// Assign an exponential value to a variable.
scientificValue = 1.34e+17;

// Assign a hex value to a variable.
aHexValue = 0xFF34;

// Assign an octal value to a variable.
AnOctalValue = 0177;

// Use an unusual variable identifier name.
$_funnyVarName = "Be careful with var names";

// Declare and initialise several variables on one line.
var aaa = 1, bbb = 2, ccc = 3;

// Store an object reference in a variable.
myObject = new Object();
```

See also:	Constant, Function arguments, Garbage collection, Memory leak, Reference counting, Scope, Scope chain

Cross-references:

ECMA 262 edition 2 – section – 7.5

ECMA 262 edition 2 – section – 10.1.3

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 3 – section – 7.6

ECMA 262 edition 3 – section – 10.1.3

ECMA 262 edition 3 – section – 12.2

O'Reilly *JavaScript Definitive Guide* – page – 52

Wrox *Instant JavaScript* – page – 13

Variable Declaration (Definition)

Variable declarations are stored as properties in an object.

Availability:	ECMAScript edition – 2

To create a variable called `myVariable`, you need to place a declaration like this in the script source text:

```
var myVariable;
```

You can optionally assign an initial value to the variable as it is declared. Like this:

```
var myVariable = "Initial value";
```

The semantics of the ECMAScript standard dictate that variable declarations must follow Function Declarations and Formal Parameter List processing.

The initial value of variables as they are declared is set to undefined. Any attributes of the property in the variable object are defined by the type of code. If a variable already exists and a declaration duplicates the same identifying name, the present value will not be replaced by the undefined value. In particular, a variable declaration cannot replace a previously declared function definition or formal parameter name.

See also:	Declaration, Definition, Formal Parameter List, `function( ... ) ...`, Script Source Text, `var`, Variable instantiation, Variable statement

Cross-references:

ECMA 262 edition 2 – section – 10.1.3

ECMA 262 edition 3 – section – 10.1.3

O'Reilly *JavaScript Definitive Guide* – page – 53

Variable instantiation (Definition)

Variables are local to one execution context.

Availability:	ECMAScript edition – 2

Every execution context has its own variable object associated with it. Variables declared in the source text are added as properties to the variable object.

This mechanism is a way of making sure that variables that are local to one execution context, do not affect the values of variables in another.

In the case of global and `eval()` code, any functions that are declared in the script source text are also added as properties of the variable object. JavaScript 1.3 and ECMAScript edition 3 introduces nested functions which means they will be added too.

For function, anonymous, and implementation-supplied code contexts, any passed in arguments (parameters) are also added as properties of the variable object.

Which particular variable object is used, and what attributes its properties take on depends on the type of code.

See also:	Execution context, Formal Parameter List, `function`(...) ..., Script Source Text, Variable Declaration

Cross-references:

ECMA 262 edition 2 – section – 10.1.3

ECMA 262 edition 3 – section – 10.1.3

Variable name (Definition)

The name of a variable that is unique within a block but not necessarily within the scope chain.

See also:	Variable, Identifier

Cross-references:

Wrox *Instant JavaScript* ISBN 1-861001-27-4 – page – 14

Variable statement (Definition)

A variable statement uses the `var` keyword to preface a list of variable declarations.

Availability:	ECMAScript edition – 2

A variable statement uses the `var` keyword to preface a list of variable declarations. The statement is terminated with a semi-colon.

You can declare the variables one at a time or several at once with the same `var` statement.

When the variables are declared, an initial value can be assigned to them at the outset.

Variables can be declared inside a function body. When they are declared there, they become local only to that function. When the function exits, the execution context is popped off the scope chain and the variables are discarded.

Variables can also be declared and assigned within a `for` loop header block.

See also:	= (Assign), Assignment expression, `delete`, Statement, `var`, Variable Declaration

Cross-references:

ECMA 262 edition 2 – section – 10.1.3

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 3 – section – 10.1.3

ECMA 262 edition 3 – section – 12.2

VBArray object (Object/JScript)

A special JScript object for interacting with Visual Basic or VBScript array data.

Availability:	JScript – 3.0 Internet Explorer- 4.0	
JavaScript syntax:	IE	*myVBArray* = new VBArray()
	IE	*myVBArray* = new VBArray(*aVisBasArray*)
	IE	*myVBArray* = *VBArray*
Argument list:	*aVisBasArray*	An array created inside Visual Basic or VBScript
Object methods:	dimensions(), getItem(), lbound(), toArray(), ubound()	

Because VBScript is only available on the Windows platform, this object has limited use in portable applications.

You might find it useful in an intranet scenario where you have total control over which browsers are deployed to the users. In that case, you know that your pages will be viewed on a compatible platform and can happily use this object with impunity.

Warnings:

- Because the Visual BASIC interpreter is only available on a limited number of platforms, this functionality is likely to be limited only to those pages that need to work on the MSIE browser in a Windows environment. This technique is unlikely to work in any other context.

Example code:

```
<!-- An example taken from Wrox: Professional JavaScript -->

<SCRIPT LANGUAGE="VBScript">
function getArray()
dim arrVB(1)
arrVB(0) = 100
arrVB(1) = 250
getArray = arrVB
End function
</SCRIPT>

<SCRIPT LANGUAGE="JavaScript">
var vbArr = new VBArray(getArray());
alert(vbArr.dimensions());
</SCRIPT>
```

Method	JavaScript	JScript	N	IE	Opera	Notes
`dimensions()`	-	3.0 +	-	4.0 +	-	-
`getItem()`	-	3.0 +	-	4.0 +	-	-
`lbound()`	-	3.0 +	-	4.0 +	-	-
`toArray()`	-	3.0 +	-	4.0 +	-	-
`ubound()`	-	3.0 +	-	4.0 +	-	-

VBArray() (Constructor)

A constructor function for creating new VBArray objects.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`VBArray object`	
JavaScript syntax:	IE	*`new VBArray(aVisBasArray)`*
Argument list:	*`aVisBasArray`*	An array created inside Visual Basic or VBScript

This constructor would be useful if you are interacting with Visual Basic scripts or functions that need to have an array passed to them.

VBArray.dimensions() (Method)

A method for requesting the number of dimensions of the array.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myVBArray`*`.dimensions()`

This tells you the number of axes in the array; you can then determine the maximum size they are expected to occupy with the `lbounds()` and `ubounds()` methods.

VBArray.getItem() (Method)

An accessor method for retrieving items from the array.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`User defined`	
JavaScript syntax:	IE	*`myVBArray`*`.getItem(anIndex)`
	IE	*`myVBArray`*`.getItem(`*`anIndex1, anIndex2, ...`*`)`
Argument list:	*`anIndex`*	A reference to an item in the array
	`anIndexN`	As many dimensions as are required to address the required item

This acessor method is used to extract the values from cells within the array. You cannot address the cells directly as you would with a native JavaScript, array although you can convert the VBArray to a native JavaScript array. Be careful though with multi-dimensional arrays.

You can use the `dimensions()` method to find out how many axes there are for addressing a multidimensional array. The `lbounds()` and `ubounds()` methods can be used to determine the extent of those axes.

VBArray.lbound() (Method)

A method that returns the index position of the first element in the VBArray.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myVBArray`*`.lbound(`*`aDimension`*`)`
Argument list:	*`aDimension`*	The dimension for which the lower bounds is required

This is the lower boundary value for the indicated dimension. You will need to measure this for each dimension individually if the array is multi-dimensional.

VBArray.toArray() (Method)

A conversion method for creating a JScript array object from a VBArray object.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Array object`	
JavaScript syntax:	IE	*`myVBArray.toArray()`*

You can convert an entire Visual Basic array to a JScript array. It is possible some data will get lost along the way but simple data formats should survive the translation.

Multi-dimensional arrays consume a great deal more memory than you anticipate. The compounded effect of adding each dimension usually increases storage by at least an order of magnitude each time.

See also:	`Array` object

VBArray.ubound() (Method)

A method that returns the index position of the last element in the VBArray.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*`myVBArray`*`.ubound(`*`aDimension`*`)`
Argument list:	*`aDimension`*	The dimension for which the upper bounds is required

This is the upper boundary value for the indicated dimension. You will need to measure this for each dimension individually if the array is multi-dimensional.

vCard object (Object/JScript)

This is an object accessible only through the user preferences interface in the MSIE browser.

Availability:	JScript – 3.0 Internet Explorer – 4.0

The `vCard` object is the parent object that contains all the user preferences settings.

There are basically three kinds of properties:

- Those that are members of the `vCard.Business` object
- Those that are members of the `vCard.Home` object
- Those that are members of the `vCard` object directly

Here is a list of all the user preference attributes that you can request:

- `vCard.Business.City`
- `vCard.Business.Country`
- `vCard.Business.Fax`
- `vCard.Business.Phone`
- `vCard.Business.State`
- `vCard.Business.StreetAddress`
- `vCard.Business.URL`
- `vCard.Business.Zipcode`
- `vCard.Cellular`
- `vCard.Company`
- `vCard.Department`
- `vCard.DisplayName`
- `vCard.Email`
- `vCard.FirstName`
- `vCard.Home.City`
- `vCard.Home.Country`
- `vCard.Home.Fax`
- `vCard.Home.Phone`
- `vCard.Home.State`
- `vCard.Home.StreetAddress`
- `vCard.Home.Zipcode`
- `vCard.Homepage`
- `vCard.JobTitle`
- `vCard.LastName`
- `vCard.MiddleName`
- `vCard.Notes`
- `vCard.Office`
- `vCard.Pager`

See also:	`Navigator.userProfile`, `userProfile.addReadRequest()`, `userProfile.getAttribute()`

Version History (Background)

Historical details of JavaScript versions.

See also:	JavaScript version, JScript version, History

view-source: URL (Request method)

You can use this for debugging in both Netscape and MSIE.

This is a useful debugging aid in some circumstances. Sometimes it is hard to download a file and this may give you a workaround.

To display a window containing the HTMLized version of a directory in the client machine, try this:

```
view-source:/
```

It works at least on Netscape Navigator version 4.7 on a Macintosh. It likely works on other platforms and versions of Netscape too.

The top level directory is the folder in which Netscape lives.

On a Macintosh at least, you can append a disk volume name, like this:

```
view-source:/MacintoshHD/
```

Beyond that you can build a path to any document in the Macintosh system, including this, which is very interesting.

```
view-source:/MacintoshHD/System%20Folder/System
```

This displays a preferences file:

```
view-source:/MacintoshHD/System%20Folder/Preferences/Fetch%20Shortcuts
```

With this level of read access to your client machine, you might be able to browse various file content that normally you wouldn't have time to do. Point at a file and its data is visible right there on the screen in a browser window.

Warnings:

- Be very careful what you browse and how. This may void your warranty. Your mileage may vary. You may corrupt your system although read-only access is unlikely to cause any harm.

See also:	`file:` URL, `javascript:` URL, URL

Visual filters (Definition)

The MSIE browser in version 4.0 and upwards now supports visual transition effects to use when modifying page content or navigating from page to page.

These filters have been enhanced and added to at version 5.0 and 5.5 of JScript. As of version 5.5 of MSIE, the performance of these filters is also optimized and enhanced.

The following kinds of visual filters are supported:

- Procedural surfaces
- Static filters
- Transitions

See also:	Procedural surfaces, Static filters, `style.filter`

void (Operator/unary)

Force an undefined value to replace an operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 2.0 Internet Explorer – 4.0 Netscape version – 3.0 Opera browser – 3.0
Property/method value type:	`Undefined primitive`
JavaScript syntax:	- `void (anExpression)` - `void anExpression`
Argument list:	*`anExpression`* An expression to be evaluated

The `void` operator is used to allow the operand to be evaluated in the normal way (perhaps it is an expression or function call), but to force an undefined value to be returned in its place.

A very useful place for this is when you create JavaScript: URLs. Making sure the result of the expression is void helps the browser cope with the fact you are calling a script and not fetching a document. Don't use `void` however if you want the result of the JavaScript execution to be used as the content of a window.

This shows how to use void in a click handler:

```
<a HREF="javascript:void(callHandler('testString'));">
```

This shows how to force JavaScript result data into a window:

```
<a HREF="javascript:'<HR>Some text here</HR>'">
```

You can also use the `void` operator to manufacture an undefined value in older browsers that have no keyword already defined. The expression (`void 0`) is just such a value. This is unnecessary now that JScript 5.5 supports an undefined value in compliance with ECMA edition 2.

The associativity is from right to left.

Refer to the operator precedence topic for details of execution order.

This keyword also represents a Java data type and the void keyword allows for the potential extension of JavaScript interfaces to access Java applet parameters and return values.

This technique is also useful if you want to evaluate an expression merely for the benefit of its side effects and without any interest in the value it returns.

Warnings:

- The `void` keyword is not available in Netscape version 2.02, or MSIE version 3.02 or earlier versions of either.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
if(document.myUndefinedProperty == (void 0))
{
   document.write("An undefined property has been referenced");
}
else
{
   document.write("A defined property was used");
}
</SCRIPT>
</BODY>
</HTML>
```

See also: Assignment expression, Associativity, `javascript`: URL, LiveConnect, Operator, Operator Precedence, typeof, Unary expression, Unary operator, `undefined`

Cross-references:

ECMA 262 edition 2 – section – 11.4.2

ECMA 262 edition 3 – section – 11.4.2

Wrox *Instant JavaScript* – page – 21

void expression (Definition)

An expression whose value is discarded and is evaluated purely for its side effects.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
function clickMe(aString)
{
   alert(aString);

   return "Some results to be ignored";
}
</SCRIPT>

<A HREF="javascript:clickMe('Test');">Test</A>
</BODY>
</HTML>
```

See also:	Constant expression, Conversion, Side effect

volatile (Reserved word)

Reserved for future language enhancements.

The addition of this operator suggests that volatile identifiers may be supported in a later version of the ECMAScript standard.

See also:	const, Reserved word

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.3

WAP (Standard)

Wireless Application Protocol.

This is a popular standard for use in mobile computing devices. It uses a derivative of the JavaScript standard language called WScript. This is constructed within a framework called WML, which is built around a structure organized like a stack of cards rather than pages, although the linkages between them are similar.

At the time of writing, this standard has become somewhat popular but there is already talk of it only being a transitory system. Future mobile computing devices may use something more sophisticated and therefore WAP may become less popular. Ultimately it will be made obsolete by a new standard.

See also:	Interpret, WML, WScript

watch() (Function/global)

Set a watch-point for a named property of an object.

Availability:	JavaScript - 1.2 Netscape - 4.0	
JavaScript syntax:	N	`myObject.watch(`*`aProperty`*`, `*`aHandler`*`)`
Argument list:	*`aHandler`*	A handler that gets called when the property changes
	`aProperty`	A property to watch

This method is provided to ease the task of debugging JavaScript.

It provides a general purpose way to call an unconnected function when a property value is changed. The function does not need to be called explicitly. The function that gets called has a particular API, which passes the following values:

- Property name
- Old value
- New value

It gets an opportunity to modify the new value or veto the change by returning the old value. Whatever value is returned is stored in the property. You can carry out other JavaScript tasks during this property call, although it is probably best to avoid making changes to other property values with watch-points that call the same handler, because you could set up a recursive loop.

If you invoke the `watch()` method without specifying a receiving object, as if it were a function, you are actually setting watch-points on global object properties. Since this is where global variables live, you can monitor them as easily as object properties.

The new event model supported by Netscape 6.0 and that already available in MSIE 5.0 present a `propertyName` property that belongs to the `Event` object. You can inspect that during an `onPropertyChanged` event and achieve the same `watch()/unwatch()` behavior.

Example code:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT>
// Code works with Netscape 4+ only
// Define initial value for property
var XXX = 10;
// Define watch handler function
function watchHandler(aProp, anOldVal, aNewVal)
{
   var myText = "";

   myText += "Property name ...: ";
   myText += aProp;
   myText += "\n";

   myText += "Old value .......: ";
   myText += anOldVal;
   myText += "\n";

   myText += "New value .......: ";
   myText += aNewVal;
   myText += "\n";

   alert(myText);
   return aNewVal;
}
// Register the watch handler
watch("XXX", watchHandler);
</SCRIPT>
Some body text
<SCRIPT>
// Modify the property to trigger the watch handler
XXX = 1000;
</SCRIPT>
</BODY>
</HTML>
```

See also:	Event, Event handler, Event management, Event model, `Event` object, `onPropertyChange`, `unwatch`(), Watchpoint handler

Cross-references:

Wrox *Instant JavaScript* –page - 56

Watchpoint handler (Interface)

The handler that is connected to a watch point has a special pre-defined API specification.

JavaScript syntax:	-	`function anId(aProp, oldVal, newVal) { someCode; return actualVal}`
Argument list:	`actualVal`	The value that will be placed into the property
	`anId`	A name for your handler function
	`aProp`	A formal parameter to pass the property name in
	`newVal`	A format parameter to pass the new value in
	`oldVal`	A formal parameter to pass the old value in

Your handler function is passed to the `watch()` method belonging to the object whose property you want to monitor.

When that property changes, your handler will be called.

You will be passed the following:

- The property name
- The old value
- The new value

Whatever you return from this handler will be stored in the property and will become its new value. This means you can return the old value, forcing the property to be read-only. You might change the new value to something else. Perhaps you would force the value to be all uppercase regardless of how it had been specified. You may even want to display an alert to warn the operator that the property is being changed.

See also:	`watch()`

Wave() (Filter/visual)

A visual filter for creating ripple effects.

Availability:	JScript - 3.0 Internet Explorer - 4.0

Refer to:

filter - `Wave()`

.web (File extension)

A compiled JavaScript and HTML application for Netscape Enterprise Server.

See also:	File extensions, Netscape Enterprise Server

Web browser (Definition)

A web browser can be used on a desktop computer, mobile device or TV set-top box to view web pages.

When JavaScript is built into a web browser, the interpreter is an integral part of the application. However, it is technically possible to separate the JavaScript interpreter and provide it as a service that is available to all applications in the system. This seems to be the way that JScript is going on the Windows platform. Whether the browser-based script environment is truly sharing the Windows Script Host facilities may depend on the browser version being used.

A web browser based interpreter will execute JavaScript that is provided in the following containers:

- An HTML page
- A JavaScript include (.js file)
- A Java archive (.jar file)

To add JavaScript to your HTML pages, you need to add `<SCRIPT>` tags as containers for the script source text.

See also:	.jar, .java, .js, `<SCRIPT>`, Host environment, HTML, HTML file, iCab, `Image` object, `Image()`, `Image.Class`, Internet Explorer, Netscape Navigator, OpenTV, Opera, Platform, WebTV

Cross-references:

Wrox *Instant JavaScript* – page - 5

Wrox *Instant JavaScript* – page - 41

Web scripting (Definition)

Web browsers provide a host environment for client-side computation.

Availability:	ECMAScript edition - 2

Web browsers provide a host environment in which to view a web page downloaded from a remote server or from a local file system. Scripts running in that browser use an object model representation of the browser and the presently viewed document. These are called the Browser Object Model and the Document Object Model respectively. There are objects to represent windows, dialog boxes, alerts, text areas, anchors, frames and all the functionality that the browser provides through its graphical user interface.

The host environment provides a means of connecting events to scripts and those scripts are then triggered when the user interacts with the document or browser. Because the entire complex provides a framework for execution and that execution is event driven, there is no `main()` function as you would have with the C language for example.

JavaScript that executes in the browser is called client-side script code. Similar script code can be executed in the web server as part of the web page request-response loop. That code would generally be executed in a more serial fashion in response to a single event (generate a page). This sort of activity is called server-side scripting.

A complete web-based application can be built with code distributed between the server and client environments.

Cross-references:

ECMA 262 edition 2 – section - 4.1

ECMA 262 edition 3 – section - 4.1

Web server (Definition)

An application that delivers web content on request from a browser.

There are many web server products available. Here are a few:

Server	Notes
Apache	At the time of writing it is currently at version 1.3.12 although minor updates happen all the time. The version 2.0 of Apache is now being seeded for beta testing.
Enterprise Server	A Netscape server product.
Fnord	A free web server that runs on Windows platforms
IIS	Internet Information Server. A Microsoft product
Intrabuilder	A Borland server product.
PWS	Personal Web Server. A Microsoft product
WebSite	A commercial PC-based web server.
WebStar	A Macintosh-based web server.

Cross-references:

Wrox *Instant JavaScript* – page - 5

WebTV (TV Set-top Box)

An analogue interactive TV set-top box.

Set-top boxes fall into several categories. From the point of view of a JavaScript developer, the most important category is the 'Browser in a box' model. This takes a basic Netscape Navigator or MSIE browser, places it inside a modest functionality PC and allows its page content to be overlaid on top of the video that is broadcast off-air. The video can also be placed into a cell within the page to allow the web page to be placed behind the video.

This is generally accomplished by allowing the `<IMG>` tag to take its source from a new URL type. Instead of an HTTP: protocol, the TV: protocol is used to trigger the video overlay hardware in place of a web request.

The WebTV boxes have been available for several years and are an analogue TV-set top box.

The JellyScript interpreter used in the WebTV box underwent an upgrade in late Spring 2000 and was released for public use during the Summer. It is generally referred to as the Summer 2000 release. You should be somewhat careful with releases of interpreters in set-top boxes, as a new release is likely to be presented for each manufacturing run. These will generally contain only minor changes. The set-top boxes are also designed to allow the interpreters to be upgraded remotely when they connect. This can lead to some interesting problems if you connect a UK PAL compatible box to an American NTSC service because the video hardware is reconfigured and is then rendered unusable.

The integration of the web content with the TV service is by means of crossover links that are URL values embedded into the video signal on a very low data rate channel that is part of the closed captioning signal.

The URL is encoded with a checksum and arrives at a rate of about 100 characters per second. Because the transfer is at such a low bit rate the URL values need to be short. When the box detects a cross-over link, it displays a small icon in the top right of the screen and the user can elect to request the associated page. The box then dials an ISP and the page is delivered in the normal way.

This is a quite good and workable enhancement to the TV service and although its deployment is limited mainly to the United States, boxes have been trialled in Europe and elsewhere.

There are a few limitations imposed on the JavaScript supported by the box and the embedded browser does not generally support extensions such as Java, plugins and ActiveX but nevertheless you can still accomplish quite a lot.

See also:	ATVEF, Interpret, JellyScript, Microsoft TV, Platform, Script execution, TV Set-top boxes, Web browser

Cross-references:

http://developer.webtv.net/authoring/javascript/javascript.htm

Week day (Time calculation)

A value between 0 and 6.

Availability:	ECMAScript edition - 2
Property/method value type:	`Number primitive`

Calculating weekdays is a simple modulo and offset of the day number derived from the time value.

The formula for calculating day number is shown here:

```
t = an instant in time measured in milliseconds relative to 01-January-1970 UTC.

msPerDay = 86400000

Day(t) = floor(t/msPerDay)

WeekDay(t) = (Day(t) + 4) modulo 7
```

The resulting values will be from 0 to 6 with Sunday being represented by 0 and Saturday by 6.

By way of proof, `WeekDay(0)` should yield 4 which represents Thursday, 01-January-1970.

Example code:

```
// Grab the time now in milliseconds
myMilliseconds = new Date().getTime();
document.write("Day number ...: ");
document.write(DayNumber(myMilliseconds));
document.write("<BR>");
document.write("Weekday number ...: ");
document.write(WeekDayNumber(myMilliseconds));
document.write("<BR>");
// Work out day number from milliseconds
function DayNumber(aMillisecondTime)
{
   msPerDay = 86400000
   myDay = Math.floor(aMillisecondTime/msPerDay);

   return myDay;
}
// Work out the week day number based on a thursday start
// This should be equivalent to Date.getDay().
function WeekDayNumber(aMillisecondTime)
{
  return ((DayNumber(aMillisecondTime) + 4) % 7);
}
```

See also:	Day number, Time range

Cross-references:

ECMA 262 edition 2 – section - 15.9.1.6

ECMA 262 edition 3 – section - 15.9.1.6

Wheel() (Filter/transition)

Reveals the new image with a rotating wheel effect.

Availability:	JScript - 5.5 Internet Explorer - 5.5

Refer to:

filter - `Wheel()`

while(...) ... (Iterator)

An iterator mechanism – a loop construct.

Availability:	ECMAScript edition - 2 JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Netscape Enterprise Server version - 2.0 Opera - 3.0	
JavaScript syntax:	-	`while ( aCondition ) { someCode }`
	-	`aLabel: while ( aCondition ) { someCode }`
Argument list:	*aCondition*	If true, the loop cycles once more
	aLabel	An optional identifier name for the loop
	someCode	The code that gets looped

Although the `while()` statement is an iterator, it is functionally related to the `if()` statement since it will execute the statement block only as long as the condition evaluates to `true`.

The difference between `if()` and `while()` is that `if()` only processes the statement block once whereas `while()` processes the statement block repeatedly until something causes the condition enclosed in parentheses to evaluate to `false`.

A while loop tests the condition before execution of each pass through the loop. If a do loop is supported by the implementations, it would test the condition after each pass through the loop.

A `break` statement can be used to terminate a `while()` iterator prematurely, perhaps within a conditional test that is supplementary to the one in the `while()` heading.

A `continue` statement can be used to initiate the next cycle of the `while()` iterator.

The unlabeled form is more commonly used and was available from earlier releases of the JavaScript and JScript interpreters. Labeling was added later at version 1.2 but is not often used.

If a labeled `continue` is used, the condition is tested again, and the loop will cycle if necessary.

Warnings:

- Make sure that something in the statement block will cause the test condition to change to `false` otherwise you will create an endless loop that can never exit. How this is dealt with depends on what you are doing in the loop and whether the implementation can detect an endless loop situation. It is likely the process containing the JavaScript interpreter will either stall and hang or a runtime error may result. In extreme cases, the hosting application may crash and on some platforms, the entire system may halt. At the very least, you could expect memory and CPU usage to go up while the loop runs. On a multi-user system, you may be able to use an administrator account to kill the offending process.

Example code:

```
// An enumerator built with a while statement
var a = 10;

while(a>0)
{
   document.write("*");
   a--;
}

document.write("<BR>");

// a labelled enumerator
a=0;

while(a<20)
{
   document.write(a);
   a++;
   if(a>10)
   {
      continue;
   }
   document.write("*<BR>");
}
```

See also: `break`, Compound statement, `continue`, `do ... while( ... )`, Flow control, `for( ... ) ...`, `for( ... in ... ) ...`, `if( ... ) ...`, Iteration statement, Label, Obfuscation, Off by one errors

Cross-references:

ECMA 262 edition 2 – section - 12.6.1

ECMA 262 edition 2 – section - 12.7

ECMA 262 edition 2 – section - 12.8

ECMA 262 edition 3 – section - 12.6.2

Wrox *Instant JavaScript* – page - 23

Wrox *Instant JavaScript* – page - 25

Whitespace (Definition)

Whitespace is used to separate tokens from one another.

Availability:	ECMAScript edition - 2

Whitespace is used to improve the readability of the script and to separate tokens from one another where they could be misinterpreted if they were concatenated together.

Whitespace characters are insignificant to the script execution apart from how they may affect the interpretation of tokens. The placement of whitespace can affect the way an expression is evaluated. For example:

```
a = 10000;
```

Will assign the value 10000 to the variable `a`, whereas:

```
a = 10 000;
```

May assign the value 10 to variable `a` but will likely generate a syntax error unless the interpreter is especially forgiving. Strictly speaking the interpreter should reject this:

```
a = c ++;
```

Is also incorrect since the postfix operator is dissociated from the variable it operates on by the whitespace in between.

Actually, there are remarkably few places that whitespace cannot be introduced.

The following characters are considered to be whitespace in ECMAScript conforming JavaScript interpreters:

Escape Sequence	Unicode Value	Name	Symbol
\t	\u0009	Tab	<TAB>
-	\u000B	Vertical Tab	<VT>
\f	\u000C	Form Feed	<FF>
-	\u0020	Space	<SP>
-	\00A0	No-break space	<NBSP>
-	Other category "Zs"	Other Unicode space characters	<USP>

ECMA edition 3 adds a couple of new whitespace character definitions. One is the non-breaking space and the other refers generally to Unicode spacing characters.

See also:	Lexical convention, Lexical element

Cross-references:

ECMA 262 edition 2 – section - 7.1

ECMA 262 edition 3 – section - 7.2

O'Reilly *JavaScript Definitive Guide* – page - 28

window (Property)

The `window` object also known as `window.window`.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*myWindow*.window
	-	window

See also:	`self,Window.frame,Window.window`

Property attributes:

`ReadOnly`.

Window adornments (Definition)

Another name for the furniture that surrounds a window.

Refer to:

Window furniture

Window events (Definition)

Some events within the event-handling complex relate to windows and their behaviors.

These events are generally triggered by clicking on a window adornment or executing a method call on a window that simulates the effect of clicking on a window border item.

See also:	`onBack`, `onForward`, `onLoad`, `onMove`, `onResize`, `onScroll`, `onUnload`

Window feature list (Definition)

The list of available features that a `window.open()` method can apply.

The feature list defines how the new window that is to be created will appear. For example, you can control the appearance of the various window adornments.

The features are generally presented as a name-value pair, thus:

```
name=value
```

The features are either available for use as simple switches and will control the functionality depending on whether they are present or not. Some also support an optional value while others require a mandatory value. Features requiring a mandatory value are generally numeric in nature.

The switching values may be "`yes`" or "`1`" to enable a feature and "`no`" or "`0`" to disable a feature. Switching features can be used with no associated value. This will generally default to "`yes`".

As you would expect, MSIE and Netscape Navigator support a completely different and incompatible set of features. Even controlling the same attribute of the window may require different feature names to be used (for example `width` in MSIE and `innerWidth/outerWidth` in Netscape Navigator).

The feature names are case insensitive but good style dictates that they are specified as shown in the table.

Here is a summary of the available features of a `window.open()` method:

Feature	Value	NNav	MSIE	Description
`alwaysLowered`	-	4	No	This dictates that a window should always be at the bottom of the stack of windows.
`alwaysRaised`	-	4	No	This dictates that a window should always be at the top of the stack of windows.
`channelMode`	-	No	4	Controls whether the window is presented in channel mode.
`dependent`	Yes/No	4	No	If a window is not dependent, it can survive after its creator has been closed.
`directories`	Yes/No	4	No	Controls the appearance of the Netscape Navigator personal toolbar.
`fullscreen`	-	No	4	On the Windows platform, MSIE will fill the screen with the window.
`height`	Number	2	4	This will set the window to the height value.
`hotkeys`	Yes/No	4	No	Setting this value to "`no`" will disable most keyboard shortcuts apart from the quit option.
`innerHeight`	Number	4	No	This will set the window to the height value measured inside the window border.
`innerWidth`	Number	4	No	This will set the window to the width value measured inside the window border.
`left`	Number	4	4	The left edge of the window will be positioned N pixels from the left edge of the screen.
`location`	Yes/No	2	4	The location bar is visible.
`menubar`	Yes/No	2	No	The menubar is fully populated with menus and menu items.
`outerHeight`	Number	4	No	This will set the window to the height value measured outside the window border.
`outerWidth`	Number	4	No	This will set the window to the width value measured outside the window border.
resizable	Yes/No	2	4	The window displays resize facilities according to the switch value.
`screenX`	Number	4	No	An alternative name for the `left` feature.
`screenY`	Number	4	No	An alternative name for the `top` feature.
`scrollbars`	Yes/No	2	4	Scroll bars are explicitly displayed according to the switch value.
`status`	Yes/No	2	4	The window displays a status bar according to the switch value.
`toolbar`	Yes/No	2	4	The window displays a toolbar according to the switch value.
`top`	Number	4	4	The top edge of the window will positioned N pixels from the top edge of the screen.
width	Number	2	4	This will set the window to the width value.
z-lock	-	4	No	This dictates that a window should always be at the same z-position in the stack of windows.

The following z-order feature switch values require the `UniversalBrowserWrite` privilege to be enabled when they are used in Netscape Navigator:

- `alwaysLowered`
- `alwaysRaised`
- `hotKeys`
- `z-lock`

Warnings:

- You must not include spaces in the feature list. If you do include spaces in the feature list attribute, the script may cause some versions of Netscape Navigator to crash horribly.
- Specifying a new window with a call to `window.open()` that has no arguments results in a window that is completely unadorned in MSIE and one that has a full complement of furniture (`location`, `toolbar`, etc.) in Netscape Navigator. Specifying even a single feature means the browser will assume all others are deactivated.
- For those features that require `UniversalBrowserWrite` privilege in Netscape Navigator, the time at which you request the feature may affect how the window is presented. This may be platform specific and Netscape Navigator may behave differently according to the UI rules and appearance of the hosting platform.
- In Netscape Navigator, on the Macintosh platform, setting the `alwaysLowered` feature will place a window at the bottom of the Z stacking order. That window will be permanently inactive as long as another window is open. It will be styled as a normal window. If the `alwaysRaised` feature is applied, the window will be placed on top and will be permanently active. In addition, the style is changed to that of a floating dialog (a Windoid in Macintosh UI parlance). A floating dialog in the Macintosh has a slimmer window border and a window title bar that is not as thick as a normal window.
- The `fullscreen` mode in the MSIE browser is not supported on the Macintosh platform.
- If you specify dimensions for the window, you must specify both the horizontal and vertical values otherwise Netscape Navigator will ignore the setting. MSIE will happily take only one of the values and provide a default for the other. Netscape Navigator requires both height and width but doesn't care what sort. You can mix `innerHeight` and `outerWidth` or `height` and `innerWidth` for example.
- Note that if the `top` or `left` values are specified on their own, the missing value will default to zero in both MSIE and Netscape Navigator.
- The `scrollbars` feature forces scroll bars to be present or not according its value in both browsers. However, scroll bars appear automatically if needed on MSIE even if the `scrollbars` feature is omitted. On Netscape Navigator, they will only be available if you explicitly ask for them.

See also:	Window furniture, `Window.open()`

Window furniture (Definition)

The various controls and scrollbars on a window border. Sometimes called window adornments or chrome.

The window furniture includes the following items:

- The location bar
- The active menu bar while the window is front-most
- The personal items bar
- Horizontal and vertical scrollbars
- The status bar at the bottom of the window
- The toolbar at the top of the window

These can all be controlled from scripts in Netscape Navigator. A script is always allowed to modify its own window with a call like this:

```
open("", "_top", aFeatureList)
```

It may require privileges to be enabled if it is going to alter another window (depending on the source of each window's documents). They can also be controlled in MSIE but only when a window is created with the `window.open()` method.

See also:	Window feature list, `Window.locationbar`, `Window.menubar`, `Window.personalbar`, `Window.scrollbars`, `Window.statusbar`, `Window.toolbar`

Window object (Object/browser)

An object representing a window or frame. This object exposes methods, properties, and events associated with it to the script.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
JavaScript syntax:	-	*myWindow* = *aFrameName*
	-	*myWindow* = frames[*anIndex*]
	-	*myWindow* = opener
	-	*myWindow* = parent
	-	*myWindow* = self
	-	*myWindow* = top
	-	*myWindow* = window
	-	*myWindow* = window.open()
	IE	*myWindow* = document.parentWindow
	IE	*myWindow* = frame

Argument list:	*anIndex*	An index to a `window` object
	aFrameName	The name of a frame in the window
Object properties:	`clientInformation, clipboardData, closed, crypto, defaultStatus, dialogArguments, dialogHeight, dialogLeft, dialogTop, dialogWidth, document, event, external, frame, frameRate, history, innerHeight, innerWidth, java, length, location, locationbar, Math, menubar, name, navigator, netscape, offScreenBuffering, opener, outerHeight, outerWidth, Packages, pageXOffset, pageYOffset, parent, personalbar, pkcs11, returnValue, screen, screenLeft, screenTop, screenX, screenY, scrollbars, secure, self, sidebar, status, statusbar, sun, toolbar, top, window`	
Object methods:	`alert(), attachEvent(), back(), blur(), clearInterval(), clearTimeout(), close(), confirm(), detachEvent(), disableExternalCapture(), enableExternalCapture(), execScript(), find(), focus(), forward(), home(), moveBy(), moveTo(), navigate(), open(), print(), prompt(), resizeBy(), resizeTo(), scroll(), scrollBy(), scrollTo(), setHotkeys(), setInterval(), setResizable(), setTimeout(), setZOptions(), showHelp(), showModalDialog(), showModelessDialog(), stop()`	
Functions:	`atob(), btoa(), captureEvents(), handleEvent(), releaseEvents(), routeEvent()`	
Event handlers:	`onAfterPrint, onBeforePrint, onBeforeUnload, onBlur, onDragDrop, onError, onFocus, onHelp, onLoad, onMouseMove, onMove, onResize, onScroll, onUnload`	
Collections:	`frames[]`	

The `window` object was introduced when JavaScript was made available at version 1.0. It has been revised several times and is likely to gain new functionality with every release.

This object is added to the scope chain as the global object when scripts are executed in a web browser. This means that the properties and methods are available without needing the `window` prefix.

In a web browser this IS the global object. Adding properties (variables) during script execution adds them to the window object for the window in which the page containing the script is loaded.

The window represents the browser container that the `document` object lives in.

Since the on-screen window persists as long as the window is open, you might think it may be a useful place to store some session state data between documents. Clearly, storing session data in a `document` object is no use if the document is going to be discarded and replaced. However, anything created by a script belonging to a window is going to get zapped when the document goes away, so you cannot store persistent values in the `window` object like that because the global object for a web page is recreated each time a page is loaded.

Storing session state data is best accomplished with a frame-set and some accessor scripts that are called within it.

Event handling support via properties containing function objects was added to `window` objects in version 1.1 of JavaScript.

Warnings:

- Be aware that if you store a reference to a `window` object and the window is closed, if you don't dispose of the reference to the window object then it cannot be garbage collected. A `window` object with no associated window is not much use unless you need to keep the object persistent due to having added some properties to it. If this is the case, then, arguably, the `window` object was the wrong place to put such things.

See also:	`BODY` object, captureEvents(), `Collection` object, `Document` object, `Document.activeElement`, `Document.captureEvents()`, `Document.frames[]`, `Document.parentWindow`, `Document.releaseEvents()`, `EventCapturer` object, `Frame` object, `Frames` object, `Global` object, `IFRAME` object, `Layer.captureEvents()`, `Layer.releaseEvents()`, `Layer.window`, `self`, `Window.frame`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
`clientInformation`	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly, DontEnum.
`clipboardData`	-	5.0 +	-	5.0 +	-	-	-	-
`closed`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning, ReadOnly.
`crypto`	1.2 +	-	4.04 +	-	-	-	-	ReadOnly.
`defaultStatus`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
`dialogArguments`	-	3.0 +	-	4.0 +	-	-	-	ReadOnly.
`dialogHeight`	-	3.0 +	-	4.0 +	-	-	-	-
`dialogLeft`	-	3.0 +	-	4.0 +	-	-	-	-
`dialogTop`	-	3.0 +	-	4.0 +	-	-	-	-
`dialogWidth`	-	3.0 +	-	4.0 +	-	-	-	-
`document`	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning, ReadOnly.
`event`	-	3.0 +	-	4.0 +	-	-	-	Warning, ReadOnly.
`external`	-	5.0 +	-	5.0 +	-	-	-	-
`frame`	-	5.0 +	-	5.0 +	-	-	-	Warning, ReadOnly.
`frameRate`	1.2 +	-	4.0 +	-	-	-	-	ReadOnly.
`history`	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning, ReadOnly.
`innerHeight`	1.2 +	-	4.0 +	-	5.0 +	-	-	Warning
`innerWidth`	1.2 +	-	4.0 +	-	5.0 +	-	-	Warning
`java`	1.1 +	-	3.0 +	-	-	-	-	ReadOnly.

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
length	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	ReadOnly.
location	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning, ReadOnly.
locationbar	1.2 +	-	4.0 +	-	5.0 +	-	-	Warning, ReadOnly.
Math	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
menubar	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly.
name	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	1 +	-	Warning
navigator	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning, ReadOnly.
netscape	1.1 +	-	3.0 +	-	-	-	-	ReadOnly.
offScreen Buffering	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
opener	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning, ReadOnly.
outerHeight	1.2 +	-	4.0 +	-	5.0 +	-	-	Warning, ReadOnly.
outerWidth	1.2 +	-	4.0 +	-	5.0 +	-	-	Warning, ReadOnly.
Packages	1.1 +	-	3.0 +	-	3.0 +	-	-	ReadOnly.
pageXOffset	1.2 +	-	4.0 +	-	5.0 +	-	-	ReadOnly.
pageYOffset	1.2 +	-	4.0 +	-	5.0 +	-	-	ReadOnly.
parent	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning, ReadOnly.
personalbar	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly.
pkcs11	1.2 +	-	4.04 +	-	-	-	-	ReadOnly.
returnValue	-	3.0 +	-	4.0 +	-	-	-	Warning
screen	1.2 +	3.0 +	4.0 +	4.0 +	5.0 +	-	-	Warning, ReadOnly.
screenLeft	-	5.0 +	-	5.0 +	-	-	-	ReadOnly.
screenTop	-	5.0 +	-	5.0 +	-	-	-	ReadOnly.
screenX	1.2 +	-	4.0 +	-	-	-	-	-
screenY	1.2 +	-	4.0 +	-	-	-	-	-
scrollbars	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly.
secure	1.2 +	-	4.0 +	-	-	-	-	ReadOnly.
self	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	ReadOnly.
sidebar	1.5 +	-	6.0 +	-	-	-	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
status	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
statusbar	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly.
sun	1.1 +	-	3.0 +	-	-	-	-	ReadOnly.
toolbar	1.2 +	-	4.0 +	-	-	-	-	Warning, ReadOnly.
top	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning, ReadOnly.
window	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning, ReadOnly.

Method	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
alert()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
attachEvent()	-	5.0 +	-	5.0 +	-	-	-	Warning
back()	1.2 +	-	4.0 +	-	-	-	-	Warning
blur()	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	1 +	-	Warning
clear Interval()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
clear Timeout()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
close()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
confirm()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	-
detachEvent()	-	5.0 +	-	5.0 +	-	-	-	-
disable External Capture()	1.2 +	-	4.0 +	-	-	-	-	Warning
enable External Capture()	1.2 +	-	4.0 +	-	-	-	-	Warning
execScript()	-	3.0 +	-	4.0 +	-	-	-	-
find()	1.2 +	-	4.0 +	-	-	-	-	-
focus()	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
forward()	1.2 +	-	4.0 +	-	-	-	-	Warning
home()	1.2 +	-	4.0 +	-	-	-	-	-
moveBy()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
moveTo()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
navigate()	-	1.0 +	-	3.02 +	-	-	-	Warning
open()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
print()	1.2 +	5.0 +	4.0 +	5.0 +	-	-	-	Warning
prompt()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
resizeBy()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
resizeTo()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
scroll()	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning, Deprecated
scrollBy()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
scrollTo()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
setHotkeys()	1.2 +	-	4.0 +	-	-	-	-	Warning
setInterval()	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
Set Resizable()	1.2 +	-	4.0 +	-	-	-	-	Warning
setTimeout()	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
setZOptions()	1.2 +	-	4.0 +	-	-	-	-	Warning
showHelp()	-	3.0 +	-	4.0 +	-	-	-	Warning
showModal Dialog()	-	3.0 +	-	4.0 +	-	-	-	Warning
showModeless Dialog()	-	5.0 +	-	5.0 +	-	-	-	-
stop()	1.2 +	-	4.0 +	-	-	-	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onAfterPrint	-	5.0 +	-	5.0 +	-	-	-	-
onBeforePrint	-	5.0 +	-	5.0 +	-	-	-	-
onBefore Unload	-	3.0 +	-	4.0 +	-	-	-	-
onBlur	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onDragDrop	1.2 +	-	4.0 +	-	-	-	-	-
onError	1.1 +	3.0 +	3.0 +	4.0 +	3.0 +	-	-	Warning
onFocus	1.0 +	3.0 +	2.0 +	4.0 +	3.0 +	-	-	Warning
onHelp	-	3.0 +	-	4.0 +	-	-	-	Warning
onLoad	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning
onMouseMove	1.2 +	3.0 +	4.0 +	4.0 +	-	-	4.0 +	Warning
onMove	1.2 +	-	4.0 +	-	-	-	-	-
onResize	1.2 +	3.0 +	4.0 +	4.0 +	-	-	-	Warning
onScroll	-	3.0 +	-	4.0 +	-	-	-	-
onUnload	1.0 +	1.0 +	2.0 +	3.02 +	3.0 +	-	-	Warning

Window.alert() (Method)

Present an alerting dialog box.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`alert(`*`aString`*`)`
	-	*`myWindow`*`.alert(`*`aString`*`)`
Argument list:	*`aString`*	Some text to display in the alert box

This presents a dialog containing the warning message and an OK button.

Note that the text that is presented in the dialog is unformatted text and you cannot use HTML in the dialog box.

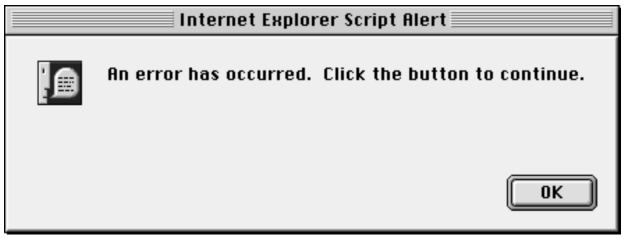
The title bar of the dialog box cannot be changed from its default setting that tells you that the dialog was invoked by JavaScript. In some browsers, it may just display the name of the browser. This is intended to stop script programmers from masquerading their dialog boxes as those of operating system diagnostics and login screens.

The `alert()` dialog box is modal on most platforms. It also blocks the script from continuing except on some versions of Netscape Navigator on Unix platforms. This behavior may become more common as desktop operating systems become more Unix-like and the Netscape Navigator core source code is deployed on newer operating systems. It is possible that this behavior will be exhibited on Mac OS X.

This method is useful for debugging. You can use it much like you would have used a `printf()` in C language debugging. Using `alert()` can sometimes be useful as an observable effect of calling a function or event handler. If you don't see the alert box, it's likely the event didn't call your handler.

You may be able to accomplish some rudimentary formatting but realistically due to font differences on platforms the only meaningful formatting is to place newline characters (`\n`) into the text to introduce a line break and to insert leading spaces for indentation.

This method does not return any meaningful value and if it is used in an assignment, the value `undefined` will be used.



Warnings:

- Be aware that the dialog may be modal but that the script may or may not continue while the `alert()` dialog is displayed. It depends on the platform. On Windows, the script execution pauses until the OK button is clicked. On UNIX, the alert is displayed by a different process or thread and the script execution continues. If you need truly modal behavior you should consider the `confirm()` or `prompt()` dialogs instead.

See also:	Debugging - client-side, Dialog boxes, `Dialog` object, `Frame` object, `Window` object, `Window.confirm()`,`Window.prompt()`

Cross-references:

Wrox *Instant JavaScript* – page - 78

Window.atob() (Function)

Decode some base-64 encoded data.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	`atob(`*`aBase64String`*`)`
	N	*`myWindow`*`.atob(`*`aBase64String`*`)`
Argument list:	*`aBase64String`*	A string containing base-64 encoded data

This function provides a means of decoding base-64 encoded values which represent an encoded form of some binary data. This encoding can be applied to text too, but is most useful where you have a block of non-textual content.

The base-64 data is decoded and converted to a block of binary data. This is then stored in a string primitive and returned to the caller as the result of the method.

To extract the binary data from the string, you will need to parse the string a character at a time and extract the numeric character value with the `String.charCodeAt()` method. You can modify the binary data directly by storing numeric values at each character position.

Note that the string will contain a sequence of 8-bit bytes and so you will need to be careful to range-limit any values that you store in the binary string.

The result is a block of binary data in a string primitive. This is somewhat cumbersome and not likely to be much used outside of a mail-reading client.

See also:	`String.charAt()`,`String.charCodeAt()`, `String.fromCharCode()`,`Window.btoa()`

Window.attachEvent() (Method)

A means of attaching events to windows and documents.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	IE	`attachEvent(`*`anEventName`*`, `*`anEventHandler`*`)`
	IE	*`myWindow`*`.attachEvent (`*`anEventName`*`, `*`anEventHandler`*`)`
Argument list:	*`anEventHandler`*	A reference to an event handler function
	`anEventName`	The name of an event to be handled

This is part of the behavior handling in MSIE which involves the use of style sheets and `.htc` files. It is a way of binding a function to an event so that when the event fires, the function is called. It can be applied in a more general way than just with behaviors.

The mechanism is quite straightforward to apply. First, create a style that can be used to attach the script to an HTML element. In that style, refer to a fragment of JavaScript contained in an external file. That external file is called an HTML Component or HTC. It is stored in an `.htc` file. The `.htc` file is invoked in a similar way to the `<SCRIPT SRC="...">` mechanism.

Then, in that `.htc` file, you create a handler script that attaches itself to whatever event you want the handler to be connected to. This `attachEvent()` method is what is used to accomplish that.

When the browser loads the page, the `.htc` file is loaded and installed and the script registers itself with the event trapping mechanisms in the browser. When the event fires, the handler script in the HTML component is executed.

There are many advantages regarding code re-use and efficiency that this technique facilitates. However, the downside is that the HTML components can result in a large number of additional requests for separate items from the web server and this can be detrimental to performance. So detrimental, in fact, that it is possible that the items may not all be loaded by the time the user is ready to interact with the page.

Two events are triggered which can be used to manage this scenario more elegantly. These are the `onContentReady` and `onDocumentReady` events. They are sent to the behavior script as a notification so you can take some action internally to prime your handler. These events would typically cause the behavior handler to set flags internally that can be checked when the handler is invoked by the user. Those flags can then lock out any interaction with the page until it is known that the content of the element and the rest of the document have been loaded. At the very least, the handler should wait until the first receipt of an `onContentReady` event.

Warnings:

- This functionality is unavailable on the Macintosh version of the MSIE browser up to at least version 5.

See also:	`<STYLE>,Document.attachEvent(), Document.detachEvent(),Window.detachEvent()`

Window.back() (Method)

A method that mimics the user clicking on the back button.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`back()`
	N	*`myWindow`*`.back()`

The browser will behave as if the user had clicked on the back button at the top of the window.

Warnings:

- If this is executed in a `<FRAMESET>` contained window, the behavior of the `Window.back()` method may not always be consistent with the `History.back()` method. One may simply affect the frame while the other may affect the entire browser window at the top of the frame-set hierarchy.

See also:	`Frame` object, `History.back()`, `Window` object, `Window.forward()`

Window.blur() (Method)

Send a `blur` event to the window object.

Availability:	DOM level - 1 JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`blur()`
	-	`myWindow.blur()`

This will take the input focus away from the receiving window, usually the one at the top of the window hierarchy. On some platforms, this will also make the window inactive and may put it to the back of the window hierarchy.

There are good arguments for using the `focus()` method on another window rather than the `blur()` method. Removing focus from a window should make it safe from inadvertent mouse clicks at the expense of making the window inactive altogether, which could be frustrating for the user.

Warnings:

- This method is not supported by MSIE version 3.
- It is possible that a window may become inactive with this method and since no window has focus you could have an active application with no active window. This is a fairly sloppy way of providing a UI interaction. A better technique is to use the `focus()` method to redirect the focus to the desired window which, by implication, `blurs()` any other that may previously have had the focus. This should result in a more consistent behavior for the user.

See also:	`Frame` object, `Input.blur()`, `Input.focus()`, `Window` object, `Window.focus()`, `Window.onblur`, `Window.onfocus`

Window.btoa() (Function)

Encode some data into base-64 form.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	N	`btoa(`*`aBinaryString`*`)`
	N	*`myWindow`*`.btoa(`*`aBinaryString`*`)`
Argument list:	*`aBinaryString`*	A string of binary data to be encoded

This function will encode the string passed as an argument and return a base-64 encoded version. Base-64 encoding is used to convert binary data into a form that survives transmission across a network, so the data passed in its input argument is really binary data. It is carried in a string value because JavaScript doesn't support a special binary container. However, JavaScript strings will happily accept 8 bit values and you can therefore store binary data in them. You would normally create the string of binary data by means of the `String.fromCharCode()` static method.

The result is a string of binary data escaped in such a way that it will survive a serial transfer through an "old-fashioned" connection. Network drivers and serial interfacing technology make this technique largely redundant and there can be few genuine applications for this other than to decode information from legacy systems or to interface to them somehow from a web browser. The functionality has very limited portability and will likely be deprecated at some stage in the future.

See also:	`String.fromCharCode()`, `Window.atob()`

Window.captureEvents() (Function)

Part of the Netscape Navigator event propagation complex.

Availability:	JavaScript - 1.2 Netscape - 4.0
Deprecated:	JavaScript - 1.5 Netscape - 6.0

Property/method value type:	undefined	
JavaScript syntax:	N	captureEvents(*anEventMask*)
	N	*myWindow*.captureEvents(*anEventMask*)
Argument list:	*anEventMask*	A mask constructed with the manifest event constants

This is part of the event management suite which allow events to be routed to handlers other than just the one that defaults to being associated with an event.

The events to be captured are signified by setting bits in a mask.

This method allows you to specify what events are to be routed to the receiving window object.

The events are specified by using the bitwise OR operator to combine the required event mask constants into a mask that defines the events you want to capture. Refer to the Event Type Constants topic for a list of the event mask values.

A limitation of this technique is that, ultimately, only 32 different kinds of events can be combined in this way and this may limit the number of events the browser can support. If you need to build complex event handling systems in Netscape Navigator 4.x, you will have implement scripts using this technique. A different script will be required for MSIE.

You may be able to factor your event handler so that you only have to make platform specific event dispatchers and can call common handling routines that can be shared between MSIE and Netscape.

This method is supported by virtually every object by virtue of the fact that it is available as a method of the `Global` object in Netscape Navigator. Therefore it gets inherited into the scope chain for every script and function (method).

Warnings:

- Since a bit mask is being used, this must be an `int32` value. This suggests that there can only be 32 different event types supported by this event propagation model.
- This capability is deprecated and is not supported in Netscape 6.0 . It has never been supported by MSIE which implements a completely different event model. As it turns out, the DOM level 2 event model converges on the MSIE technique.

Example code:

```
// Build and setup a mask for several events
myEventMask = Event.KEYDOWN | Event.MOUSEDOWN | Event.RESET;
window.captureEvents(myEventMask);
function EventHandler(anEventObject){//... some event handling code here}
window.onkeydown   = EventHandler;
window.onmousedown = EventHandler;
window.onreset     = EventHandler;
```

See also:	`captureEvents()`, `Document.captureEvents()`, `Element.onevent`, Event propagation, Event type constants, `Frame` object, `Layer.captureEvents()`, `onMouseMove`, `Window` object, `Window.releaseEvents()`

Window.clearInterval() (Method)

Cancel a previous `setInterval()` timer that caused a function to be called periodically.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`clearInterval(`*`anIntervalID`*`)`
	-	*`myWindow`*`.clearInterval(`*`anIntervalID`*`)`
Argument list:	*`anIntervalID`*	The ID of an interval returned by the `setInterval()` method

The interval timer mechanism can be used to repeat the execution of a block of script code at regular intervals. You can establish several blocks of repeating code at once if necessary, so there could be a number of pending interval timers. You must be able to identify the one that you want to clear.

When you create an interval timer with the `setInterval()` method, it will return back to you with a value that corresponds uniquely to that interval's set-up context. You can use that value later to clear the interval timer.

You must be careful not to clear an interval more than once.

Warnings:

- Be careful when clearing interval timers. If you try to clear one that does not exist, it can sometimes crash the browser.

See also:	`Frame` object, Interval handlers, Timeout handlers, Timer events, `Window` object, `Window.clearTimeout()`, `Window.setInterval()`, `Window.setTimeout()`

Window.clearTimeout() (Method)

Clear a previously established timeout function call.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0

Property/method value type:	undefined	
JavaScript syntax:	-	clearTimeout(aTimeoutID)
	-	*myWindow*.clearTimeout(*aTimeoutID*)
Argument list:	*aTimeoutID*	The ID of a timeout returned by the setTimeout() method

The timeout mechanism can be used to defer the execution of a block of script code. You can defer several blocks at once if necessary, so there could be a number of pending timeout triggers. You must be able to identify the one that you want to clear.

When you create a timeout trigger with the setTimeout() method, it will return back to you with a value that corresponds uniquely to that trigger's set-up context. You can use that value later to clear the timeout trigger.

You must be careful not to clear timeout triggers more than once.

Warnings:

- Be careful when clearing interval timers. If you try to clear one that does not exist, it can sometimes crash the browser.
- It is highly likely that the interval may have elapsed and the timeout no longer exists to be cleared. A more reliable technique that is less prone to browser crashes is to set a flag value in a global variable and then test that from within the timeout invoked function.

See also:	clearTimeout(), Frame object, Timeout handlers, Timer events, Window object, Window.clearInterval(), Window.setTimeout()

Window.clientInformation (Property)

This is another more appropriate name for the navigator object.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	Navigator object	
JavaScript syntax:	IE	clientInformation
	IE	*myWindow*.clientInformation

This is an alternative name for the window.navigator property.

Warnings:

- Note that the clientInformation reference is another name for the navigator object. We could be uncharitable and suggest that this is an attempt by Microsoft to avoid publicising a competitor's browser name in any scripts that they publish. It is likely that scripts written by and for Microsoft products would use this syntax rather than the navigator property, and will break on Netscape Navigator.

See also:	`Frame` object, `Window` object, `Window.navigator`

Property attributes:

`ReadOnly`, `DontEnum`.

Window.clipboardData (Property)

An object containing data that represents the contents of the clipboard.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
Property/method value type:	`clipboardData object`	
JavaScript syntax:	IE	`clipboardData`
	IE	*`myWindow`*`.clipboardData`

If you want to move data in and out of the clipboard on a Windows platform from within the MSIE browser, this property will yield a reference to a `clipboardData` object that encapsulates the clipboard contents.

Example code:

```
// Example of how to use the clipboard object supplied by
// Martin Honnen.
clipboardData.setData('Text', 'All for Kibology');
alert(clipboardData.getData('Text'));
clipboardData.clearData();
alert(clipboardData.getData('Text'));
```

See also:	`clipboardData` object

Window.close() (Method)

This will close the window.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`close()`
	-	*`myWindow`*`.close()`

The method `window.close()` will attempt to close the window in which the script is executing. This is only possible without a user confirmation dialog if the script has the `UniversalBrowserWrite` privilege. The statement `self.close()` is effectively `window.close()` and will also attempt to close the window in which the script is executing.

A JavaScript running inside a window may close that window.

JavaScripts running outside a window may or may not be able to close that window. This may be implementation dependent and may also depend on the value of the `opener` property of the target window and the current security policy in force. Attempts to close windows that your JavaScript did not open will elicit a user dialog to get permission to close the window. This is so that people cannot write invasive scripts that wreak havoc on your browser session when they are loaded.

The default behavior is to allow JavaScript to close windows which were opened by scripts that were originally served from the same server and which were not opened by the user. This may be overridden if additional privileges are granted to scripts.

From version 1.1 of JavaScript, (generally speaking) you can only close windows that were opened by JavaScript from within a script. Although, certain browser privileges will allow you to close other windows and this behavior may be browser dependent.

Warnings:

- Do not confuse this method with the `document.close()` method. They are not the same.
- Do not call this for `Window` objects that represent frames, you cannot close a single frame within a window.

See also:	`Frame` object, `Frame.close()`, `UniversalBrowserAccess`, `UniversalBrowserWrite`, `Window` object, `Window.open()`, `Window.opener`

Window.closed (Property)

A property value that is `true` if the window is closed.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`closed`
	-	`myWindow.closed`

From JavaScript version 1.1 onwards, a `window` object may persist after it has been closed. This may seem odd but it is possible that the `window` object will have had other object references added to it and for it to be deleted, it must relinquish those references. Without this object persistence, a dangerous cascading delete effect may happen, inadvertently discarding all manner of objects within your script context.

The only valid thing you should access at this point is the `closed` property. This allows you to check for the window having been closed by the user before you try to do something with it.

Some programmers would argue that this is poor technique anyway and that a reference to a window held externally should be nulled by the closing function. That way you can test a variable that exists within the script's scope chain rather than an object that should probably have been purged from memory. Given the bugs in the implementation of this feature, that is a better way to determine whether a window is still in existence.

Warnings:

- This value can be read by an unsigned script in another window. However, Netscape Navigator (at least on the Macintosh platform) exhibits some bugs with this whole mechanism and the property value is of limited use in any case.
- For a start, you cannot access the `closed` property of the object that the `Window.open()` method returns because it is an `EventCapturer` object and not a `window` object. You can contrive to store a reference to the new window in the original window's properties by making it pass a reference to its `window.self` property which needs to be stored in a property belonging to its `window.opener`. However, if the child window is then closed, the object that refers to it becomes void and there is no access to any of its properties, let alone its `window.closed` property.
- There are sufficient problems in this area that without some quite tricky scripting you cannot make use of this facility in a portable manner.

See also:	`Frame` object, `Window` object

Property attributes:

`ReadOnly`.

Window.confirm() (Method)

Present a confirmation dialog box.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0
Property/method value type:	`Boolean primitive`

JavaScript syntax:	-	`myResult = confirm(aString)`
	-	`myResult = myWindow.confirm(aString)`
Argument list:	`aString`	Some text to explain what is to be confirmed

This presents a modal dialog containing the confirmation message and two buttons, OK and Cancel. This is useful because you often need confirmation from a user.

This method is useful for debugging. An example showing how it can be used for debugging a recursive function is given below.

Note that the text that is presented in the dialog is plain unformatted text and you cannot use HTML text in the dialog box.

The title bar of the dialog box cannot be changed from its default setting which tells you that the dialog was invoked by JavaScript. In some browsers, it may just display the name of the browser. This is intended to stop script programmers from masquerading their dialog boxes as those of operating system diagnostics and login screens.

The `confirm()` dialog box is modal and blocking. The script must wait for a response from the user.

When the user clicks on either of the buttons, the result returned by the method indicates which one was chosen. The `true` value indicates the OK button was clicked and `false` indicates the Cancel button.

Example code:

```
// Example showing the use of a confirm() dialog to
// debug recursive calls provided by Martin Honnen.
function showTime()
{
   if (confirm('Time is: ' + new Date() + '. Show again?'))
   {
      setTimeout('showTime()', 1000 * 5);
   }
}
showTime();
```

See also:	Debugging - client-side, Dialog boxes, `Dialog` object, `Frame` object, `Window` object, `Window.alert()`,`Window.prompt()`

Cross-references:

Wrox *Instant JavaScript* – page - 78

Window.crypto (Property)

A reference to a `Crypto` object for security encoding.

Availability:	JavaScript - 1.2 Netscape - 4.04	
Property/method value type:	`Crypto object`	
JavaScript syntax:	N	`crypto`
	N	*`myWindow`*`.crypto`

The object referred to by this property is used with the browser security model. Refer to the topic that discusses the `Crypto` object for more details.

See also:	`Crypto` object

Property attributes:

`ReadOnly`.

Window.defaultStatus (Property)

A property containing the text displayed in the status bar.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	`defaultStatus`
	-	`defaultStatus = `*`aString`*
	-	*`myWindow`*`.defaultStatus`
	-	*`myWindow`*`.defaultStatus = `*`aString`*
Argument list:	*`aString`*	A new value for the default status

As the mouse rolls over active elements in the page, they may write values into the status bar. This is the value that is restored when the mouse rolls out of the object.

Although this is called the `defaultStatus` value, you can change its setting by writing a new value into the property.

A useful technique is to reset this value to an empty string using a `<BODY>` or `<FRAMESET>` `onUnload` handler.

As an example of a way to use this property, it may be useful to give the user some helpful message when loading a <FORM> into the window.

You can read or write the value in this property.

Because this also works with frames, you can set the value for this to be different according to the frame that the mouse is over. That way, the message can give the user some context sensitive hints on a frame by frame basis. You need to be careful about this because if you define the defaultStatus value in a top level window that contains frames, when the mouse is in the frames, the defaultStatus value will be that which is defined for the frame and not the window. The window's default status will only be displayed when the mouse is over the border in between the frames, hence you must define the defaultStatus value for all frames as well as the window.

If you only need to display a temporary message, then use the status property instead of the defaultStatus property.

Warnings:

- The default status value may not be properly restored on some platforms. Macintosh and X-Windows versions of Netscape 3 may exhibit this problem as may other browser and platform combinations.

See also:	Frame object, onMouseOut, onMouseOver, Status line, Window object, Window.status

Window.detachEvent() (Method)

A means of detaching events from windows and documents that were previously attached with the attachEvent() method.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
JavaScript syntax:	IE	detachEvent(*anEventName*)
	IE	*myWindow*.detachEvent(*anEventName*)
Argument list:	*anEventName*	The name of an event to be handled

This is part of the behavior handling in MSIE which involves the use of style sheets and .htc files.

See also:	<STYLE>, Document.attachEvent(), Document.detachEvent(), Window.attachEvent()

Window.dialogArguments (Property)

The arguments passed to a model dialog in a `showModalDialog()` call.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`dialogArguments`
	IE	*`myWindow`*`.dialogArguments`

This is only available when the window is contained within a modal dialog. It is the value contained in the second argument of the `showModalDialog()` method. You can use this to pass values into the modal dialog and the `returnValue` property of the window object inside the modal dialog to return a value to the caller.

Refer to the `Window.showModalDialog()` topic for an example of how this works.

See also:	`Window.showModalDialog()`

Property attributes:

`ReadOnly`.

Window.dialogHeight (Property)

The height of a modal or modeless dialog window.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`dialogHeight`
	IE	*`myWindow`*`.dialogHeight`

Dialog windows are a special kind of window supported by MSIE, principally to provide a more sophisticated alternative to an `alert()`, `confirm()` or `prompt()` dialog.

This property yields the height of such a dialog window which can be changed by assigning a value to this property from a script running in the window.

The value returned will be a value and a measurement unit in the same style as would be used with a CSS positioning style property. This means it will be a string and not a numeric value and will need to be parsed carefully.

Window.dialogLeft (Property)

The left edge of a modal or modeless dialog window.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`dialogLeft`
	IE	*`myWindow`*`.dialogLeft`

Dialog windows are a special kind of window supported by MSIE, principally to provide a more sophisticated alternative to an `alert()`, `confirm()` or `prompt()` dialog.

This property yields the position of the left edge of such a dialog window which can be changed by assigning a value to this property from a script running in the window.

The value returned will be a value and a measurement unit in the same style as would be used with a CSS positioning style property. This means it will be a string and not a numeric value and will need to be parsed carefully.

Window.dialogTop (Property)

The top edge of a modal or modeless dialog window.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`dialogTop`
	IE	*`myWindow`*`.dialogTop`

Dialog windows are a special kind of window supported by MSIE, principally to provide a more sophisticated alternative to an `alert()`, `confirm()` or `prompt()` dialog.

This property yields the position of the top edge of such a dialog window which can be changed by assigning a value to this property from a script running in the window.

The value returned will be a value and a measurement unit in the same style as would be used with a CSS positioning style property. This means it will be a string and not a numeric value and will need to be parsed carefully.

Window.dialogWidth (Property)

The width of a modal or modeless dialog window.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`dialogWidth`
	IE	*`myWindow`*`.dialogWidth`

Dialog windows are a special kind of window supported by MSIE, principally to provide a more sophisticated alternative to an `alert()`, `confirm()` or `prompt()` dialog.

This property yields the width of such a dialog window which can be changed by assigning a value to this property from a script running in the window.

The value returned will be a value and a measurement unit in the same style as would be used with a CSS positioning style property. This means it will be a string and not a numeric value and will need to be parsed carefully.

Window.disableExternalCapture() (Method)

Part of the Netscape Navigator 4 event propagation complex.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`disableExternalCapture()`
	N	*`myWindow`*`.disableExternalCapture()`

This method allows you to inhibit the receipt of events from other window objects that might have been loaded from a server other than the one that provided the script it is called from. You need higher than normal privilege to execute this method. However this method itself is quite harmless, the one that really requires the security interlocking is the `window.enableExternalCapture()` method.

Warnings:

- Although this method can be called without any special privilege being necessary, your script will need to be granted the `UniversalBrowserWrite` privilege to enable the external capture again.

See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.enableExternalCapture()`

Window.document (Property)

A reference to the document object that is contained in the window.

Availability:	DOM level - 1 JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Document object`	
JavaScript syntax:	-	`document`
	-	*`myWindow`*`.document`

This property returns a reference to an object that represents the document that is currently loaded into the window.

Warnings:

- Beware of recursion effects by referring to the document belonging to the window and then referring back unintentionally via a property enumeration. The `document.parentWindow` property in MSIE will come straight back to the source window.
- Beware that this refers to an object that belongs to the window and not the object that the window belongs to.

See also:	`Document` object, `Document.parentWindow`, `Element.document`, `Frame` object, `Window` object

Property attributes:

`ReadOnly`.

Cross-references:

Wrox *Instant JavaScript* – page - 80

Window.enableExternalCapture() (Method)

Part of the Netscape Navigator 4 event propagation complex.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`enableExternalCapture()`
	-	*`myWindow`*`.enableExternalCapture()`

This method allows you to begin capturing events from other window objects that might have been loaded from a server other than the one that provided the script it is called from. This is not usually permitted because you could build a script that watches for users typing a password, for example. You need higher than normal privileges to execute this method.

Warnings:

- This method requires your script to be granted the `UniversalBrowserWrite` privilege to operate properly in Netscape Navigator.

See also:	`disableExternalCapture()`, `Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.disableExternalCapture()`

Window.event (Property)

During event handling, MSIE stores a reference to an event object in this variable.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`Event object`	
JavaScript syntax:	IE	`event`
	IE	*`myWindow`*`.event`

In MSIE, this property yields a reference to an event object. This only has any meaning during an event handler call and it is shared by all events which you would assume will cause some problems when multiple events are triggered. However, it seems to sort itself out most of the time. You will most likely refer to the event property as a member of the `Global` object, omitting the `window` prefix. The handler can acquire the object if it needs it to find out more about the context of the event.

In Netscape Navigator prior to version 6.0, the event object is passed as an argument to the event handler function. Netscape 6.0 implements the DOM level 2 event model which works like the MSIE technique.

This means your event handlers must be written slightly differently if they need to access an event object that pertains to the event itself.

To get a roughly portable handler together, the MSIE event can be stored in a local variable. Then, as it turns out, it is easier to convert the old style Netscape Navigator properties to MSIE compatible properties. You may not need to convert all of them and you may need to do some experimentation to see that the bit-masking of modifier keys is correctly set up for your target browser.

The `Closure()` object technique applies here when assigning function objects to be event handlers although that is not supported by MSIE version 4.

This entire event managing complex is under review and some work on standardization happens at DOM level 2 with more to come in DOM level 3. We may have to experience several more browser revisions before we can rely on a truly portable approach and that won't solve legacy browser issues until everyone has upgraded.

Warnings:

- The MSIE event-handling model is radically different to the Netscape Navigator event-handling model. Unless you are doing only very simple event handling you will likely need to code for both models and somehow make them both available in the same page.

Property attributes:

`ReadOnly`.

Window.execScript() (Method)

Execute a script on behalf of a window.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`User defined`	
JavaScript syntax:	IE	`execScript(`*`aSourceText`*`)`
	IE	`execScript(`*`aSourceText`*`, `*`aLanguage`*`)`
	IE	`myWindow.execScript(`*`aSourceText`*`)`
	IE	`myWindow.execScript(`*`aSourceText`*`, `*`aLanguage`*`)`
Argument list:	*`aSourceText`*	Some legal script source text
	`aLanguage`	The language to execute the script source in

This is somewhat like an `eval()` call except that the script runs in the context and scope chain of the target window and not the window whose script makes the call.

Because this is currently only supported in MSIE, you may not find it very useful.

See also:	`Frame` object, `Window` object

Window.external (Property)

Reference to an external object outside of the interpreter.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
Property/method value type:	`External object`	
JavaScript syntax:	IE	`external`
	IE	*`myWindow`*`.external`

The MSIE browser supports a means of access to the surrounding object space that contains the browser and the desktop environment in which it executes. This of course is only available on the Windows platform and its use is discouraged for reasons of portability.

See also:	external

Window.find() (Method)

This duplicates the behavior of the FIND button on the Netscape Navigator button bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Deprecated:	Netscape - 6.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	`find()`
	-	`find(`*`aSearchKey`*`)`
	-	`find(`*`aSearchKey`*`, `*`aCaseSense`*`)`
	-	`find(`*`aSearchKey`*`, `*`aCaseSense`*`, `*`aDirection`*`)`
	-	*`myWindow`*`.find()`
	-	*`myWindow`*`.find(`*`aSearchKey`*`)`
	-	*`myWindow`*`.find(`*`aSearchKey`*`, `*`aCaseSense`*`)`
	-	*`myWindow`*`.find(`*`aSearchKey`*`, `*`aCaseSense`*`, `*`aDirection`*`)`
Argument list:	*`aCaseSense`*	A switch for case sensitivity
	`aDirection`	A direction to search
	`aSearchKey`	The text to search for

All of the arguments are optional. However, you must specify the first argument if want to specify the second and so on.

If no arguments are specified, a dialog box is presented to the user for them to specify the search attributes.

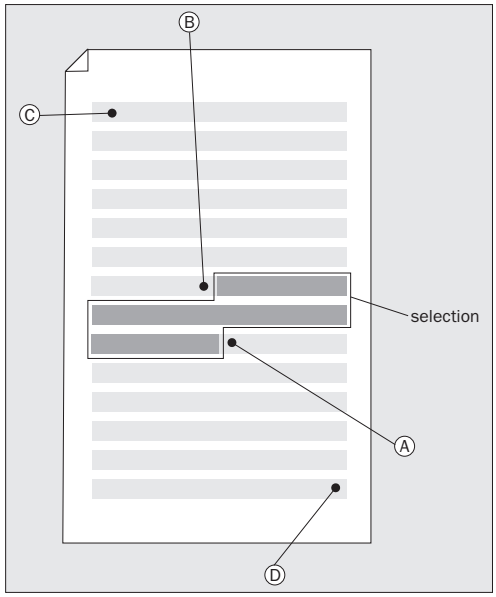
The search key is a arbitrary string of characters. The search facility will look for this string in the current page.

The case sensitive switch parameter must be `true` to force a case sensitive search and `false` to ignore case. By default a case insensitive search is carried out if this argument is omitted.

The direction switch is a `true` for a backwards search and `false` for a forwards search. The default is a forwards search if this argument is omitted.

Forward searches commence at the current cursor position or immediately after the current selection if there is one, see position A on figure overleaf. If there is no selection or the cursor has not been focussed into the page and positioned there, then the search begins at the top of the document, see position C on figure overleaf. In the case of a backwards direction, the search starts immediately in front of the selection if there is one , see position B on figure overleaf, or the end of the document if not, see position D on figure overleaf.

The result is `true` if the text was found in the page and `false` if it was absent.



See also:	`Frame` object, `Window` object, `Window.home()`, `Window.print()`, `Window.stop()`

Window.focus() (Method)

Send a `focus` event to the window object.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`focus()`
	-	*`myWindow`*`.focus()`

This restores focus to the receiving window or frame that the method is executed on. If the focus goes to a frame, then by implication, the window it belongs to becomes active too.

On some platforms this will bring the receiving window to the foreground if it is not already the active window. It is generally better technique to use the `focus()` method on another window rather than call the `blur()` method if you want to remove focus from a window. However, that may not always be possible if you want to completely remove focus from all windows. The consequence may be that the window becomes inactive and on some platforms it means the window will be placed at the rear of any other windows on the screen.

Warnings:

- This method is not supported by MSIE version 3.

See also:	`Frame` object, `Input.blur()`, Input.focus(), `Window` object, `Window.blur()`, `Window.onblur`, `Window.onfocus`

Window.forward() (Method)

Mimics the effect of the user clicking on the FORWARD button.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`forward()`
	N	*`myWindow`*`.forward()`

The browser will behave as if the user had clicked on the FORWARD navigation button.

Warnings:

- If this is executed in a `<FRAMESET>` contained window, the behavior of the `window.forward()` method may not always be consistent with the `history.forward()` method. One may simply affect the frame while the other may affect the entire browser window at the top of the frame-set hierarchy.

See also:	`Frame` object, `History.forward()`, `Window` object, `Window.back()`

Window.frame (Property)

This is another name for `self` and `window`.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
Property/method value type:	`Window object`	
JavaScript syntax:	IE	`frame`
	IE	`myWindow.frame`

This property yields an object that represents the frame that contains this `window` object. This is the same value that the `window` property yields. It is another gratuitous extension that MSIE provides so that scripts to run inside frames can look more consistent with what they are doing and yet break on any non-MSIE browsers due to the frame property not being present.

`Frame` objects support all the methods and properties of `window` objects although they may not always be meaningful in the context of a frame living in a frame-set.

Of course, you can emulate this in other browsers by assigning the value of the `window.window` property to the `window.frame` property anyway.

This appears to be undocumented and does not appear in the Microsoft reference information. Nevertheless, the property is visible by enumeration of the `window` object and appears to work.

Warnings:

- Using this property renders your scripts non-portable unless you code around the missing property on non-MSIE browsers.

See also:	`Document.parentWindow`, `self`, `Window` object, `Window.self`, `Window.top`, `Window.window`

Property attributes:

`ReadOnly`.

Window.frameRate (Property)

An indication of the frame rate for the current display.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	`frameRate`
	N	*`myWindow`*`.frameRate`

This property was discovered accidentally by enumerating the properties of a window object in the Netscape Navigator browser.

The name of this property suggests that it should yield the rate at which the display is refreshed. This should be a constant value and you would expect this to be somewhere between 50 and 85.

Property attributes:

`ReadOnly`.

Window.frames[] (Collection)

An array containing `window` objects, each one referring to the content of a frame within the window.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0

Property/method value type:	Frames object	
JavaScript syntax:	-	frames
	-	*myWindow*.frames
HTML syntax:	<FRAME><IFRAME>	

The frames property yields an array containing objects, each one of which refers to a separate frame. You can count how many there are with the window.frames.length property. In MSIE, this also includes any inline floating frames created with an <IFRAME> tag.

You can cross-reference between frames in a window by means of the frames property. Every window refers to a frames array, which contains a list of frames within that window. Each frame contains a different window. This can get confusing, but it simply means that frames correspond to windows at the basic object level.

You can also use the opener, parent and top properties as well when you are cross-referencing between windows and frames.

Frames arrays may be nested where windows contain frames within frames. The frames array is like all others, its first element is at index 0. This means that something as confusing as this is legal:

```
frames[0].frames[2].frames[1]
```

As a window reference, that looks in the current window for the first frame, then into the third frame within that and then the second frame within that.

You can use the parent window reference to access windows that are all at the same hierarchical level as the current one. This would be accomplished like this:

```
parent.frames[ ... ]
```

Warnings:

- Be careful not to confuse this with the document.frames property supported by MSIE. That is intended just to list the inline frames within a document.
- In Netscape Navigator the window.frames property points back at the window object and the frame objects and frames.length property are stored there as global variables. This is arguably a bug although not all commentators agree. The MSIE implementation is much neater however in that environment, all elements having an ID HTML tag attribute are reflected as member properties of the window. This is not correct either since they should be members of the document and not the window.
- Strangely enough, it all seems to work from the scripting point of view. You can access the length value to count the frames and the individual frame objects are available associatively by name from the window.frames property.

See also:	<MAP TARGET="...">, Document object, Document.frames[], Frame object, Frames object, Window object, Window.opener, Window.parent, Window.top

Property attributes:

ReadOnly.

Cross-references:

Wrox *Instant JavaScript* – page - 80

Window.handleEvent() (Function)

Pass an event to the appropriate handler for the window.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	undefined	
JavaScript syntax:	N	handleEvent(*anEvent*)
	N	*myWindow*.handleEvent(*anEvent*)
Argument list:	*anEvent*	An event to be handled by this object

This applies to Netscape Navigator prior to version 6.0. From that release onwards, event management follows the guidelines in the DOM level 2 event specification.

On receipt of a call to this method, the receiving object will look at its available set of event handler functions and pass the event to an appropriately mapped handler function. It is essentially an event dispatcher that is granular down to the object level.

The argument value is an event object that contains information about the event.

See also:	Event object, Event propagation, Frame object, handleEvent(), SubmitButton.handleEvent(), TEXTAREA.handleEvent(), TextCell.handleEvent(), Window object, Window.routeEvent()

Window.history (Property)

This property returns a `history` object for this window.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`History object`	
JavaScript syntax:	-	`history`
	-	`myWindow.history`

This property yields a reference to a `history` object for the session in this window. The `history` object is somewhat like an array of history items. Netscape Navigator supports a slightly more sophisticated `history` object than MSIE.

On MSIE, you can access methods belonging to the `history` object with a construct like this:

```
window.history.go(0);
```

Nevertheless, you cannot access member properties of the `history` object to examine URL values as this contradicts the security requirements.

Warnings:

- This appears to be only partially supported on MSIE for Macintosh.

See also:	`Frame` object, `History` object, `Window` object

Property attributes:

`ReadOnly`.

Window.home() (Method)

This duplicates the behavior of the HOME button on the Netscape Navigator button bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`home()`
	N	*`myWindow`*`.home()`

The result of calling this method depends on what you have set up as your home page in the browser. By default, it is likely to be a page on the web site belonging to the browser manufacturer. Commonly though, it will be a blank page because after you have been irritated by your browser automatically dialling in to your ISP, you'll likely have told it not to go to any home page by default.

See also:	`Frame object`, `Window` object, `Window.find()`, `Window.print()`, `Window.stop()`

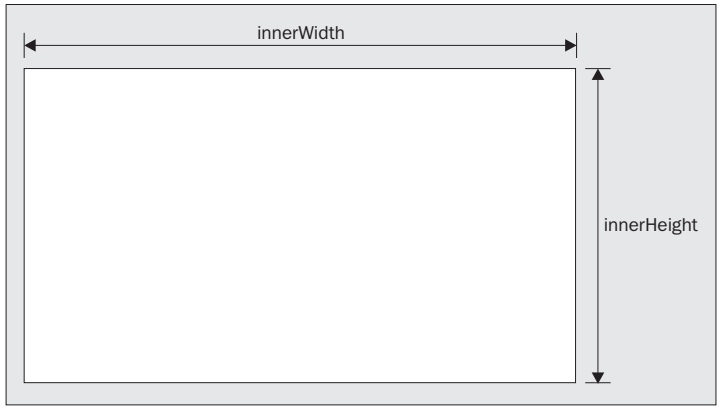
Window.innerHeight (Property)

The height of the window inside the frame.

Availability:	JavaScript - 1.2 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`innerHeight`
	-	*`myWindow`*`.innerHeight`

This property is supported on Netscape Navigator and tells you what the current height of the content area of the window is set to.

Assigning a value to this property will resize the window on Netscape Navigator. On MSIE it will be ignored.



Warnings:

- This method requires your script to be granted the `UniversalBrowserWrite` privilege to set a window size smaller than 100x100 pixels.

See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.innerWidth`, `Window.outerHeight`, `Window.outerWidth`

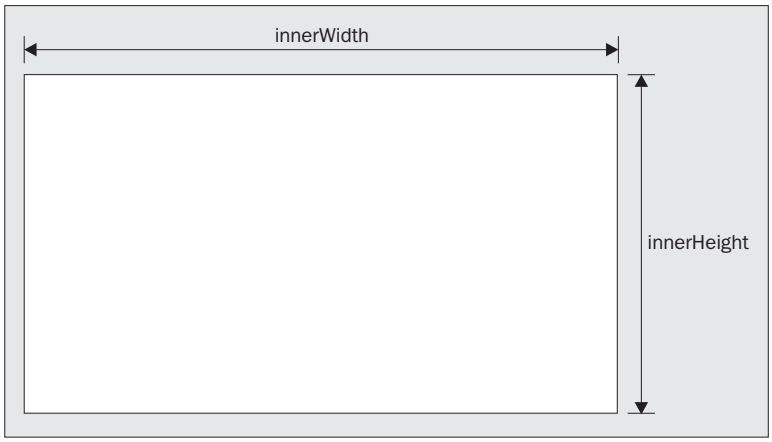
Window.innerWidth (Property)

The width of the window inside the frame.

Availability:	JavaScript - 1.2 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`innerWidth`
	-	*myWindow*`.innerWidth`

This property is supported on Netscape Navigator and tells you what the current width of the content area of the window is set to.

Assigning a value to this property will resize the window on Netscape Navigator. On MSIE it will be ignored.



Warnings:

- This method requires your script to be granted the `UniversalBrowserWrite` privilege to set a window size smaller than 100x100 pixels.

See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.innerHeight`, `Window.outerHeight`, `Window.outerWidth`

Window.java (Property)

A reference to the Java package object that is the root of the '`java.*`' packages tree.

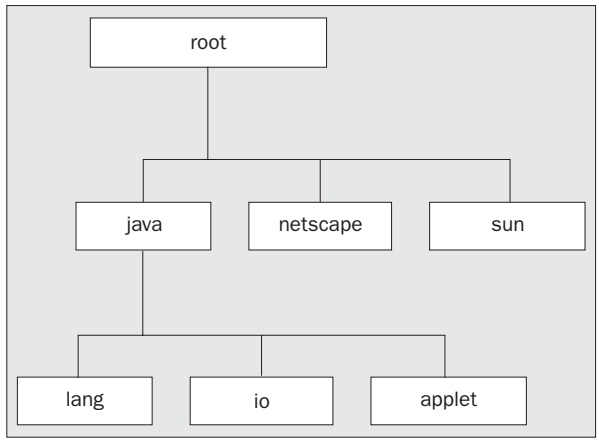
Availability:	JavaScript - 1.1 Netscape - 3.0	
Property/method value type:	`JavaPackage java`	
JavaScript syntax:	N	`java`
	N	*`myWindow`*`.java`
	N	*`myWindow`*`.Packages.java`
	N	`Packages.java`

The object referred to by this property sits at the top of the java package name hierarchy. It is through this property that you can access the java objects, properties and methods via LiveConnect.

This shortcut reference corresponds to a directory hierarchy where Java class, packages are stored. Thus the `java.lang.String class` lives in a file called `java/lang/String.class` which on some systems may be stored inside a ZIP archive.

The main shortcoming in this whole mechanism is that the browser cannot tell whether a reference to an object is a request for a JavaPackage or a JavaClass. It will assume you mean a package by default and if you misspell a class name, it won't tell you it couldn't find it.

The objects and classes supported by this access to the underlying Java engine cover a very wide topic base. We have examined only the top level functionality to try and establish points of connection between the two environments. For a full and in-depth reference coverage of the Java language environment, consult the Wrox book *"Java Programmer's Reference" by Grant Palmer*.



See also:	`JavaPackage` object, LiveConnect, `Packages.java`, `Window.netscape`, `Window.Packages`, `Window.sun`

Property attributes:

`ReadOnly`.

Window.length (Property)

The number of frames in the window.

Availability:	JavaScript - 1.0 JScript - 3.0 Internet Explorer - 4.0 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	`frames.length`
	-	`length`
	-	*`myWindow`*`.frames.length`
	-	*`myWindow`*`.length`

The same value as the `window.frames.length` property.

This property is maintained in MSIE for consistency with Netscape Navigator. The frames in MSIE are contained in a `FrameArray` object, but this does not prevent their names polluting the property namespace of the `Window` object.

See also:	`Collection.length`, `Frame` object, `length`, `Window` object

Property attributes:

`ReadOnly`.

Window.location (Property)

A reference to the `location` object that represents the URL of the current window content.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Location object`	
JavaScript syntax:	-	`location`
	-	*`myWindow`*`.location`

The attributes of the object referred to by this property can be read or written to cause the window content to be reloaded under script control.

The object itself ought to be considered read only but its properties are modifiable. Consult the document for the `location` object for details of what you can read or write. In particular, refer to the `window.location.href` value as a way to load a new document into the window.

Warnings:

- Do not confuse this location object with that belonging to the document. They are different.
- Older versions of the web browsers used to treat this property as a read/write string containing the URL. The access to the URL value is now via the properties of the object referred to by this property and not via the property itself.

See also:	`Document.location`, `Document.referrer`, `Document.URL`, `Frame` object, `Location` object, `Window` object, `Window.navigate()`

Property attributes:

`ReadOnly`.

Window.locationbar (Property)

A reference to an object that represents the location bar.

Availability:	JavaScript - 1.2 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Bar object`	
JavaScript syntax:	-	`locationbar`
	-	*`myWindow`*`.locationbar`

This is a read only property containing a reference to a `Bar` object whose `visible` property contains a Boolean value that controls the visibility of the screen furniture represented by the object.

In this case, it is supposed to control the visibility of the location bar.

The example shows how this can be done in a secure way that requires privilege to be granted and in a non-secure way that does not.

Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to change the visibility of the locationbar.

Example code:

```
// Request necessary
privilegesnetscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrit
e");
// Hide location barwindow.locationbar.visible = false;
// There is another way that works without requesting privilegewindow.open('',
'_top', 'location=0');
```

See also:	`Bar` object, `Frame` object, `UniversalBrowserWrite`, Window furniture, `Window` object, `Window.menubar`, `Window.personalbar`, `Window.scrollbars`, `Window.statusbar`, `Window.toolbar`

Property attributes:

`ReadOnly`.

Window.menubar (Property)

A reference to an object that represents the menu bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`Bar object`	
JavaScript syntax:	-	`menubar`
	-	`myWindow.menubar`

This is a read-only property containing a reference to a `Bar` object whose `visible` property contains a Boolean value that controls the visibility of the screen furniture represented by the object.

In this case, it is supposed to control the visibility of the menu bar.

The example shows how this can be done in a secure way that requires privilege to be granted and in a non-secure way that does not.

Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to change the visibility of the menubar.
- The Macintosh version of Netscape Navigator requires that the window be made inactive and then active again after the status of this item has changed for there to be any noticeable effect.
- Even then, not all items in the menubar are hidden but most are.
- Be careful if you set the menubar and toolbar to invisible, you will then have no reload capability to be able to refresh the screen. You will then only be able to quit.

- On some platforms, there may still be a refresh capability available on the contextual menu inside the window. This will depend on whether the contextual menu is still available or whether it has been changed to remove the refresh item.

Example code:

```
// Request necessary privileges
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
// Hide menu bar
window.menubar.visible = false;

// There is another way that works without requesting privilege
// window.open('', '_top', 'menubar=0');
```

See also:	`Bar` object, `Frame` object, `UniversalBrowserWrite`, Window furniture, `Window` object, `Window.locationbar`, `Window.personalbar`, `Window.scrollbars`, `Window.statusbar`, `Window.toolbar`

Property attributes:

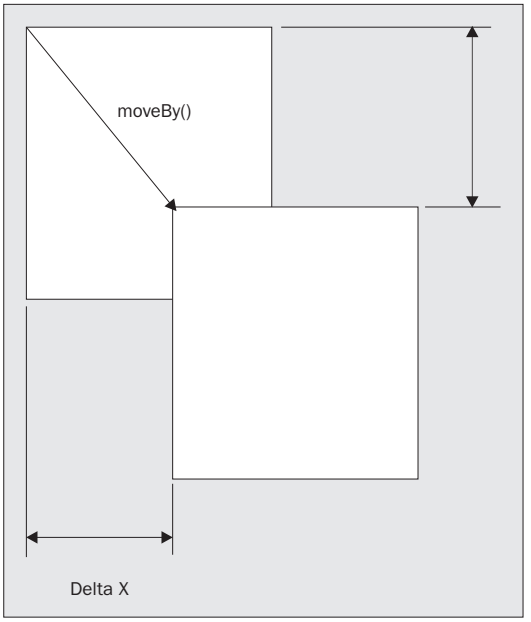
`ReadOnly`.

Window.moveBy() (Method)

Move the window by a specified distance.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`moveBy(`*`anOffsetX`*`, `*`anOffsetY`*`)`
	-	*`myWindow`*`.moveBy(`*`anOffsetX`*`, `*`anOffsetY`*`)`
Argument list:	*`anOffsetX`*	A distance in pixels
	`anOffsetY`	A distance in pixels

This method will translate a window across the screen. Positive values move the window to the right and down while negative values move the window to the left and up.



Warnings:

- There are some security implications for moving windows in Netscape 4. They are intended to stop you hiding a window by shifting it off-screen. Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to move the window outside the screen area.

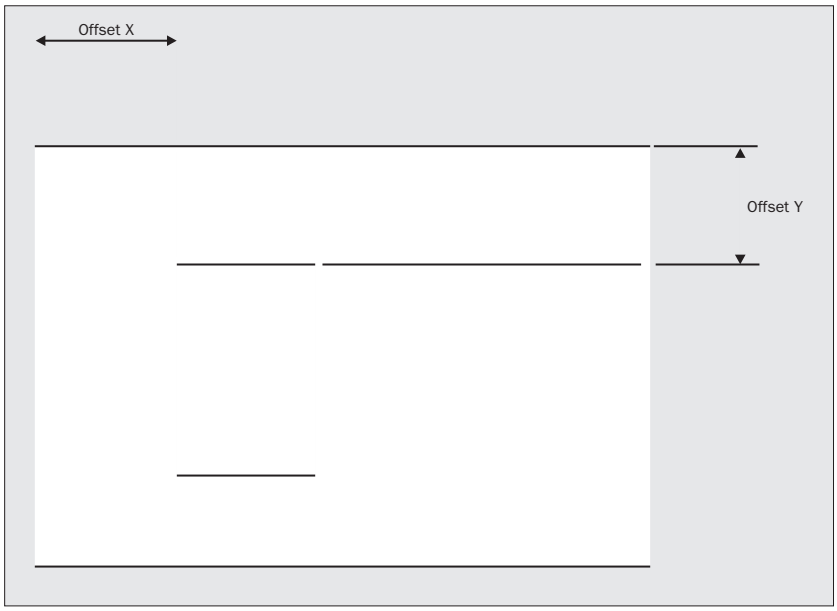
See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.moveTo()`, `Window.onmove`, `Window.pageXOffset`, `Window.pageYOffset`, `Window.resizeBy()`, `Window.resizeTo()`, `Window.scrollBy()`

Window.moveTo() (Method)

Move the window to a specific location.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`moveTo(aCoordX, aCoordY)`
	-	`myWindow.moveTo(aCoordX, aCoordY)`
Argument list:	*`aCoordX`*	A position in pixels
	`aCoordY`	A position in pixels

This method will locate a window at a specific position on screen.



Warnings:

- There are some security implications for moving windows in Netscape 4. They are meant to stop you hiding a window by shifting it off-screen. Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to move the window outside the screen area.

See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.moveTo()`, `Window.onmove`, `Window.pageXOffset`, `Window.pageYOffset`, `Window.resizeBy()`, `Window.resizeTo()`, `Window.scrollBy()`

Window.name (Property)

The name of the window either from the `<FRAME>` tag, the `Window.open()` method call or an assignment to this property.

Availability:	DOM level - 1 JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0
Property/method value type:	`String primitive`

JavaScript syntax:	-	*myWindow*.name
	-	*myWindow*.name = *aString*
	-	name
	-	name = *aString*
	-	window.open(*aURL*, *aName*, ...)
HTML syntax:	<FRAME NAME="...">Window.open(...)	
Argument list:	*aName*	A name for the window
	aString	A string value containing the new name for the window
	aURL	A URL to load into the window

This name value can be used with the target attribute of the HTML <A> anchor tag or the <FORM> tag.

At version 1.0 of JavaScript this is a read-only property.

JavaScript version 1.1 introduces the capability of changing the name of a window and makes this property writable. This is particularly useful because the initial window has no name and cannot be targeted directly until it has. You can fix this with an onLoad handler.

The name of a window can be set when the window is created with the open() method. It is also assigned by the NAME="..." HTML tag attribute of a <FRAME> tag.

The example shows the window name being modified. Reload to see the persistence effect. Load another document and then load this script again to see how the name persists through intermediate document loads.

Warnings:

- This is not the title text for the window.
- Note that the name value persists through document loads and so you may need to be careful if you are using the name property in a document script. It might have been set by an earlier page. This also suggests that other window properties may be persistent between documents and might provide a way to pass messages between pages and maintain state during a session.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
document.write("***"+window.name+"<BR>");
window.name = "FRED";
document.write("***"+window.name+"<BR>");
</SCRIPT>
</BODY>
</HTML>
```

See also:	`Frame` object, `IFRAME.name`, `NAME="..."`, `Window` object, `Window.open()`

Window.navigate() (Method)

Load a new URL into the window.

Availability:	JScript - 1.0 Internet Explorer - 3.02	
Property/method value type:	`undefined`	
JavaScript syntax:	IE	`myWindow.navigate(`*`aURL`*`)`
	IE	`navigate(`*`aURL`*`)`
Argument list:	*`aURL`*	A new location to navigate the window to

This is functionally similar to assigning a new value to the `window.location.href` property. Because it is only available in MSIE you should avoid using this in cross-browser development.

Warnings:

- Because this is only supported in MSIE, you should not use it if you want to develop portable scripts. A more portable technique is to assign the URL value to the `href` property of the `location` object that is accessible from the `window.location` property.
- `window.navigate("index.html");` and `window.location.href = "index.html";` are functionally equivalent and guaranteed to be portable between Netscape Navigator and MSIE if not across all available browsers.

Example code:

```
// Reload the window with a new relative URL valuewindow.navigate("_Window.html");
```

See also:	`Location` object, `Window.location`

Window.navigator (Property)

A reference to a navigator object that describes the browser.

Availability:	JavaScript - 1.0 JScript - 3.0 Internet Explorer - 4.0 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Navigator object`	
JavaScript syntax:	-	*`myWindow`*`.navigator`
	-	`navigator`

The `navigator` object was originally provided by Netscape Navigator as a way of making client properties available to scripts.

Warnings:

- Microsoft have property named `clientInformation` that points at the same object. This then allows people to write scripts that are functionally identical but which will break on Netscape Navigator and Opera, although perhaps we should give them the benefit of the doubt because really this property should have been called `clientInformation` in the first place.

See also:	`Frame` object, `Navigator` object, `Window` object, `Window.clientInformation`

Property attributes:

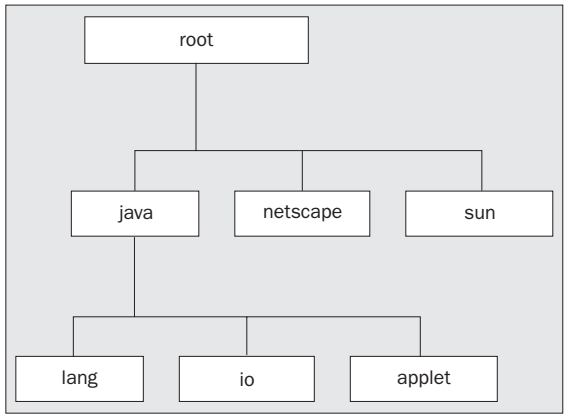
`ReadOnly`.

Window.netscape (Property)

A reference to the Java package object that is the root of the '`netscape.*`' Packages tree.

Availability:	JavaScript - 1.1 Netscape - 3.0	
Property/method value type:	`JavaPackage netscape`	
JavaScript syntax:	N	*myWindow*`.netscape`
	N	*myWindow*`.Packages.netscape`
	N	`netscape`
	N	`Packages.netscape`

The object referred to by this property sits at the top of the Netscape package name hierarchy. It is through this property that you can access the java objects, properties and methods via LiveConnect.



See also:	JavaScript to Java values, `Packages.netscape`, `Window.java`, `Window.Packages`, `Window.sun`

Property attributes:

`ReadOnly`.

Window.offscreenBuffering (Property)

A property that controls off-screen buffering effects.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`Boolean or String primitive`	
JavaScript syntax:	-	*myWindow*`.offScreenBuffering`
	-	*myWindow*`.offScreenBuffering = `*aSetting*
	-	`offScreenBuffering`
	-	`offScreenBuffering = `*aSetting*
Argument list:	*aSetting*	A new value to control this functionality

This property controls how screen updates are accomplished. This is especially useful with DHTML-based animation. An off-screen buffer is used to draw the new window contents and then copy it over the existing content when it is complete. This is much nicer to look at than the effects you get with a non-buffered redraw which clears the screen first and then draws the entire page in front of you.

The available settings are:

Boolean `true` - Activate off-screen buffering

Boolean `false` - Deactivate off-screen buffering

String "`auto`" - Let the browser decide for itself

In general, MSIE accomplishes slightly more attractive redraws with this facility enabled than Netscape Navigator. However, new versions of Mozilla 5 and Netscape 6 may surpass the MSIE version 5 support.

Warnings:

- Enabling this facility will slightly decrease CPU performance and consume more memory since a copy of the onscreen window needs to be maintained in the application memory.

See also:	`Frame` object, `Window` object

Window.onblur (Property)

This is called when the window loses input focus.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myWindow*.onblur
	-	*myWindow*.onblur = *aHandler*
	-	onblur
	-	onblur = *aHandler*
HTML syntax:	<BODY onBlur="*aHandler*"><FRAMESET onBlur="*aHandler*">	
Argument list:	*aHandler*	An event handler function object

A `blur` event is caused by the user clicking on another window or frame or the `window.blur()` method being called. When this event is triggered, an `onblur` event handler will be invoked.

The `onblur` event handler is a function which is represented by an object that is referred to by this property. Because it is stored in a property, you can change the handler by storing a reference to a different function object in this property. At least, you can on MSIE.

You cannot redefine the value of the `window.onblur` property from inside the `onblur` function handler. This means you can't modify the `onblur` behavior while a `blur` event is in progress.

The handler is registered either by assigning a function to the `onblur` property or by defining it with an HTML tag attribute.

Warnings:

- Do not use `<BODY ONBLUR="window.focus()">` to make sure the window is always on top. This does not work as you expect. The problem is caused because the `BODY` receives a `blur` event when any element inside the page takes the focus away from it (for example, when you click on a link or form element). The page becomes unusable because every attempt to click elements in the page triggers a javascript call that draws the focus straight back to the `BODY`/ window. So all you get is a window that permanently blocks all the other windows but does not allow any elements inside it to be used.
- In Netscape Navigator, the property is not enumerable if the value is defined with an `onBlur="..."` HTML tag attribute. However, it is enumerable if the property is assigned within JavaScript without there being a defining tag attribute. There may be a general rule that script defined properties are always enumerable, while internally created properties may not be.
- Displaying an `alert()`,`confirm()` or `prompt()` dialog takes focus away from a window. Don't forget that your `blur` handler will be called if this happens.
- The Macintosh version 4.7 of Netscape Navigator does not appear to support `blur` events.

See also:	onBlur,Window.blur(),Window.focus(), Window.onfocus

Property attributes:

DontEnum.

Window.ondragdrop (Property)

This event handler is called when an object is dropped into the browser window.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	Function object	
JavaScript syntax:	-	*myWindow*.ondragdrop
	-	*myWindow*.ondragdrop = *aHandler*
	-	ondragdrop
	-	ondragdrop = *aHandler*
HTML syntax:	<BODY onDragDrop="*aHandler*"><FRAMESET onDragDrop="*aHandler*">	
Argument list:	*aHandler*	An event handler function object

This event handler is invoked when the user drags an item onto a window in Netscape Navigator.

To access the details of the entity that has been dragged into and dropped on the window, you need to access the data property of the event object that is passed as an argument to the handler when it is called.

The data is a single URL when a single entity is dropped into the window or an array of strings, each containing an URL, when a collection of entities are dropped onto a window.

You will need UniversalBrowserRead privilege to access this data.

The handler is registered either by assigning a function to the ondragdrop property or by defining it with an HTML tag attribute.

Warnings:

- Note that this does not appear to work in Netscape 4.7 for Macintosh.

See also:	Event.data,onDragDrop

Property attributes:

DontEnum.

Window.onerror (Property)

A reference to an error event handler for this window object.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myWindow*`.onerror`
	-	*myWindow*`.onerror = `*aHandler*
	-	`onerror`
	-	`onerror = `*aHandler*
Argument list:	*aHandler*	An event handler function object

You can register an error handler function by assigning a function object to the window's `onerror` property so that when an error occurs in the JavaScript within the window, that function will be called.

This can be a useful way of trapping and completely inhibiting the display of error messages to the user. When the error handling function is called, it is passed three arguments:

- The textual message that explains the error
- The URL of the document that contains the error
- The line number within that document where the error occurred

The line number is the physical line number within the document, not a line number within a script, so you should be able to open the document in a text editor and go to the indicated line and identify the problem.

The only way to activate this error handling capability is to assign a function to the `onerror` property. There is no HTML tag attribute mechanism for defining an error handler for a window although you can associate one with the `<IMG>` tag with an attribute value.

You can return either `true` or `false` as a result of calling your handler:

`true` - Inhibit any further error processing and abort the script.

`false` - Hand control back to the browser to deal with the error in the normal way.

Deactivate error handling altogether by calling a function that simply returns `true`. To restore default error handling to the window later, assign a function handler to the property that merely returns a `false` value as its result without intervening in the error process in any way.

Warnings:

- Beware of the return values. Returning `true` from an error handler inhibits the browser from carrying out any further action. This is exactly opposite to the return value from a form element event handler, which requires that a `false` value be returned to inhibit any further action by the browser.
- This works on version 4 of Netscape Navigator. It is also fully supported by MSIE version 5. Version 4 of MSIE for Macintosh (and possibly other platforms) allows the error to be trapped but does not pass any meaningful argument values. They are all undefined.
- This is not supported on the WebTV platform.
- There is a problem with this event handler hook in the Netscape 6.0 browser. The values are not passed to the handler correctly although the handler is called.

See also:	Debugging - client-side, Error events, Error handler, JellyScript, `onError`

Property attributes:

`DontEnum`.

Window.onfocus (Property)

This event handler is called when a window gains the input focus.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myWindow*`.onfocus`
	-	*myWindow*`.onfocus = `*aHandler*
	-	`onfocus`
	-	`onfocus = `*aHandler*
HTML syntax:	`<BODY onFocus="`*aHandler*`"><FRAMESET onFocus="`*aHandler*`">`	
Argument list:	*aHandler*	An event handler function object

A `focus` event is caused by the user clicking on the receiving window or frame or the `window.focus()` method being called. When this event is triggered, an `onfocus` event handler will be invoked.

The `onfocus` event handler is a function which is represented by an object that is referred to by this property. Because it is stored in a property, you can change the handler by storing a reference to a different function object in this property. At least, you can on MSIE.

You cannot redefine the value of the `window.onfocus` property from inside the `onfocus` function handler. This means you can't modify the `onfocus` behavior while a `focus` event is in progress.

The handler is registered either by assigning a function to the `onfocus` property or by defining it with an HTML tag attribute.

Warnings:

- Displaying an `alert()`,`confirm()` or `prompt()` dialog may take focus away from a window on some platforms and browsers. Don't forget that your `focus` handler could be called when the dialog is dismissed if this happens.
- The Macintosh version 4.7 of Netscape Navigator does not appear to support `focus` events.

See also:	`onFocus`,`Window.blur()`,`Window.focus()`, `Window.onblur`

Property attributes:

`DontEnum`.

Window.onload (Property)

A reference to a loading completed event handler for this window object.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myWindow*`.onload`
	-	*myWindow*`.onload = `*aHandler*
	-	`onload`
	-	`onload = `*aHandler*
HTML syntax:	`<BODY onLoad="`*aHandler*`"><FRAMESET onLoad="`*aHandler*`">`	
Argument list:	*aHandler*	An event handler function object

This is called when a `<BODY>` or `<FRAMESET>` has completed loading. This event is generated when all document parsing is done, all script blocks are fully operational and all functions defined and accessible. The Document Object Model should have been fully constructed by this time.

The `onload` event handler is a function which is represented by an object that is referred to by this property. Because it is stored in a property, you can change the handler by storing a reference to a different function object in this property. However, because this is called when loading is completed and not again after that, you can logically only modify this value during page loading.

The handler is registered either by assigning a function to the `onfocus` property or by defining it with an HTML tag attribute.

Warnings:

- There are certain things you cannot do until this point in the page loading process. For example, you cannot use inline `document.write()` methods to construct an `<OBJECT>` embed in MSIE. You can however enclose a dummy `<OBJECT>` block inside a `<DIV>` and then replace the `innerHTML` of that `<DIV>` with a script generated `<OBJECT>` block that is provided in an `onload` event handler. There are some issues about how tightly that plugin is bound to the browser window. It is possible that a plugin that plays video may not track window drags with live updates to the screen quite as elegantly as an `<OBJECT>` block that is manifestly constant and not defined by a script.
- There is a bug in the way that the `onload` event handlers are invoked in Netscape Navigator 2. Logically, one would assume that the `onload` handler for the `<FRAMESET>` would be called once all of the individual frames within it have been fully loaded. However, Netscape 2 calls this prematurely and so you should make sure that you have a way to check that every one of your frames has loaded individually.

See also:	`onLoad`, `Window.onunload`

Property attributes:

`DontEnum`.

Window.onmove (Property)

This event handler is called when a window is moved.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myWindow*.onmove
	-	*myWindow*.onmove = *aHandler*
	-	onmove
	-	onmove = *aHandler*
HTML syntax:	`<BODY onMove="`*aHandler*`"><FRAMESET onMove="`*aHandler*`">`	
Argument list:	*aHandler*	An event handler function object

This is called when a window is moved on the screen to another location.

The handler is registered by defining it with an HTML tag attribute.

The event is also triggered when a window is moved under control of a script with the `moveTo()` or `moveBy()` methods.

Warnings:

- This is not supported by Netscape Navigator 4 on the Unix platform.
- On the Macintosh platform, Netscape Navigator calls the `onmove` handler twice when the window is moved.

See also:	`onMove`,`Window.moveBy()`,`Window.moveTo()`

Property attributes:

`DontEnum`.

Window.onresize (Property)

A reference to a resized window event handler for this `window` object.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myWindow*.onresize
	-	*myWindow*.onresize = *aHandler*
	-	onresize
	-	onresize = *aHandler*
HTML syntax:	`<BODY onResize="`*aHandler*`"><FRAMESET onResize="`*aHandler*`">`	
Argument list:	*aHandler*	An event handler function object

This is called when a window is enlarged or reduced in size.

The handler is registered by defining it with an HTML tag attribute. The handler can also be registered by assigning the function object to the `onresize` property of the window.

The event is also triggered when a window is resized under control of a script with the `resizeTo()` or `moveBy()` methods.

See also:	`onResize`,`Window.resizeBy()`,`Window.resizeTo()`

Property attributes:

`DontEnum`.

Window.onunload (Property)

This event handler is called when a document is cleared from a window.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Function object`	
JavaScript syntax:	-	*myWindow*.onunload
	-	*myWindow*.onunload = *aHandler*
	-	onunload
	-	onunload = *aHandler*
HTML syntax:	`<BODY onUnload="`*aHandler*`"><FRAMESET onUnload="`*aHandler*`">`	
Argument list:	*aHandler*	An event handler function object

This is called when a `<BODY>` or `<FRAMESET>` is about to be replaced by some new content.

The `onunload` event handler is a function which is represented by an object that is referred to by this property. Because it is stored in a property, you can change the handler by storing a reference to a different function object in this property.

The handler is registered either by assigning a function to the `onfocus` property or by defining it with an HTML tag attribute.

In a frame-set collection, the individual onunload handlers for each frame will be called in turn before the `onunload` event handler for the `<FRAMESET>` is called.

Use of this event handler is discouraged on the WebTV platform. There are limitations on its availability at certain times during alert and refresh actions.

See also:	JellyScript, `onLoad`, `onUnload`, `Window.onload`

Property attributes:

`DontEnum`.

Window.open() (Method)

A means of creating new windows under script control.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*aNewWindow* = *myWindow*.open()
	-	*aNewWindow* = *myWindow*.open(*aURL*)
	-	*aNewWindow* = *myWindow*.open(*aURL*, *aName*)
	-	*aNewWindow* = *myWindow*.open(*aURL*, *aName*, *aFeatureList*)
	-	*aNewWindow* = *myWindow*.open(*aURL*, *aName*, *aFeatureList*, *aFlag*)
	-	*aNewWindow* = open()
	-	*aNewWindow* = open(*aURL*)
	-	*aNewWindow* = open(*aURL*, *aName*)
	-	*aNewWindow* = open(*aURL*, *aName*, *aFeatureList*)
	-	*aNewWindow* = open(*aURL*, *aName*, *aFeatureList*, *aFlag*)
Argument list:	*aFeatureList*	A list of attributes for the new window
	aFlag	A flag to indicate how the history list is to be modified
	aName	The name of a new or existing target window
	aURL	A URL to load into the window

All arguments are optional. However, since they are also positional, you must use commas and empty strings as necessary to null out the entries you don't require if you need the later ones.

The URL value, if specified, will fetch and load the document into the window as the `window.open()` method is executed. You may omit or null the URL with an empty string. This will open a new window but not load anything into it.

The name identifies the target window. If that window name is not already used, then a new window will be created, otherwise the URL will replace the existing content. When you direct a null URL to an already open window being referenced by name just returns a reference to that `window` object. This may be useful if you know the name of a window but do not have a handle on its object representation. If you do not specify a name, then a new unnamed window will be created.

The feature list describes the attributes of the window. This may in some implementations allow you to change a window's appearance but this may not always be possible. The feature list must be comma separated and must not contain any spaces.

The flag value indicates what you want done with the history table for this window. Passing the Boolean `true` value indicates that the new URL should replace the existing history entry. A Boolean `false` value indicates that the new URL should be added to the tail of the history list. This facility is available from JavaScript version 1.1 onwards.

The value returned by this `open()` method is a reference to the `window` object that represents the window that has just been opened. You can store this in a variable so you can send messages to the window when necessary.

See the Window features list topic for a list of the window feature names and the values that they expect, and a discussion on the limitations and some subtle catch-outs and differences between the browsers.

This method returns a reference to the `window` object for the window that was created or updated. You should note this in some persistent variable if you plan to communicate with the window some time later during the session.

Warnings:

- Do not confuse the `window.open()` method with the `document.open()` method. It is probably a safer technique to explicitly refer to the `window.open()` method rather than rely on the implication of the `open()` method being sent to a `window` object. It might be possible to modify the scope chain to make this default to `document.open()` instead.
- A `window.open()` will create a new viewport window. A `document.open()` will clear the current window contents and start writing new HTML into it.
- If you include spaces in the feature list attribute, the script may cause some versions of Netscape Navigator to crash horribly and in others it will just ignore any features following the space character.
- Netscape Navigator returns an `EventCapturer` object instead of a `Window` object.
- The `window.open()` method will behave differently in JellyScript on a WebTV set-top box when compared with the behavior in a normal computer based web browser. It creates a pseudo-window in an `<IFRAME>` and appends it to the end of the current display. The user can then scroll down to this window.

See also:	`Document.open()`, `EventCapturer object`, `Frame object`, JellyScript, Window feature list, `Window object`, `Window.close()`, `Window.name`, `Window.opener`

Cross-references:

Wrox *Instant JavaScript* – page - 74

Window.opener (Property)

A reference to the window that contained the link that opened this one.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	`myWindow.opener`
	-	`opener`

This allows you to build a hierarchy of parent and child windows. A window has a `null` value for this property if the window was opened directly by the user.

If necessary, you can modify the value in this property since it supports read and write access.

Warnings:

- Not supported by Netscape 2 or version 3 of MSIE. You could simulate this property by storing the self property of the opening window in a property of your own that belongs to the object representing the new window. You could also check for the existence of this property and create it yourself if it is missing. Refer to the example to see how you can do this.
- This value is meaningful for a top level window that contains a frame-set but it is not meaningful for frames within that window. Logically, if this is to indicate a window parentage, it should also refer to a top level window and not a frame within one, even if the `window.open()` method was executed from within a frame.
- Although the value is apparently writable, it is difficult to think of a circumstance where you would want to redefine the parent window object.

Example code:

```
// Create the new window
var myNewWindow = open('http://www.wrox.com', 'windowName');
// Now set its opener property if it doesn't already exist
if (!myNewWindow.opener)
{
myNewWindow.opener = window;
}
```

See also:	Browser version compatibility, `Document` object, `Frame` object, `Window` object, `Window.close()`, `Window.frames[]`, `Window.open()`

Property attributes:

`ReadOnly`.

Window.outerHeight (Property)

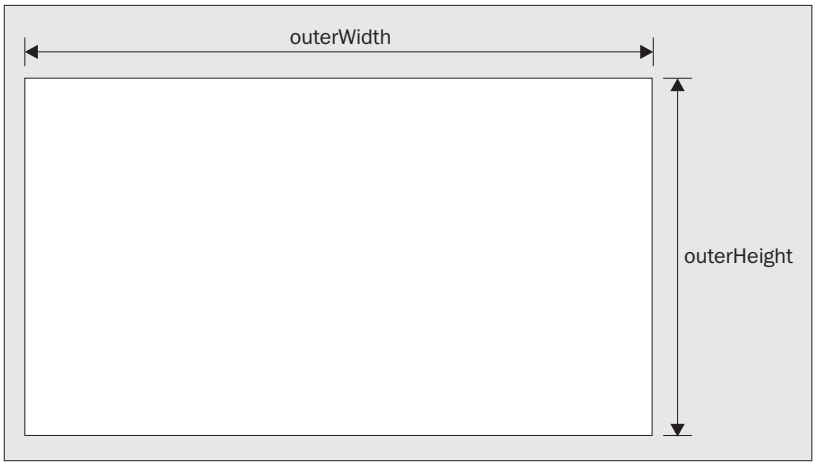
The height of the window including the frame.

Availability:	JavaScript - 1.2 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myWindow*`.outerHeight`
	-	`outerHeight`

This property is supported on Netscape Navigator and tells you what the current height of the outer bordered area of the window is set to.

Assigning a value to this property will resize the window on Netscape Navigator. On MSIE it will be ignored.

Note that the content window will be much smaller than the bordered area. The content area can be controlled directly with the `innerHeight` and `innerWidth` properties.



Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to set the window size to less than 100x100 pixels.

See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.innerHeight`, `Window.innerWidth`, `Window.outerWidth`

Window.outerWidth (Property)

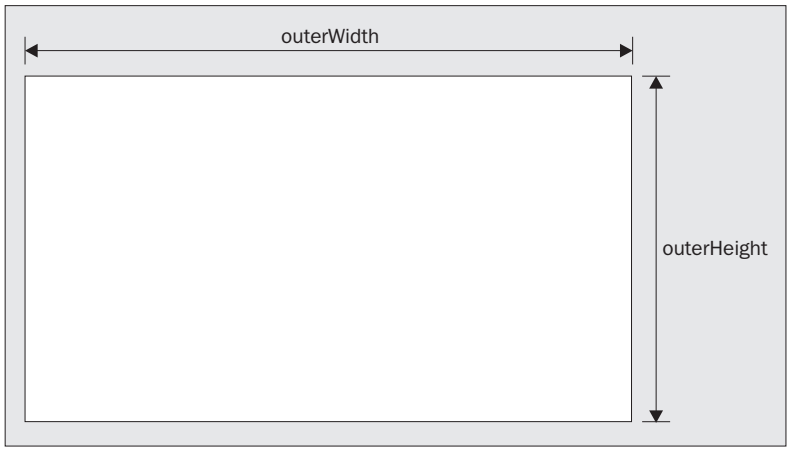
The width of the window including the frame.

Availability:	JavaScript - 1.2 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myWindow*.outerWidth
	-	outerWidth

This property is supported on Netscape Navigator and tells you what the current width of the outer bordered area of the window is set to.

Assigning a value to this property will resize the window on Netscape Navigator. On MSIE it will be ignored.

Note that the content window will be much smaller than the bordered area. The content area can be controlled directly with the innerHeight and innerWidth properties.



Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to set the window size to less than 100x100 pixels.

See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.innerHeight`, `Window.innerWidth`, `Window.outerHeight`

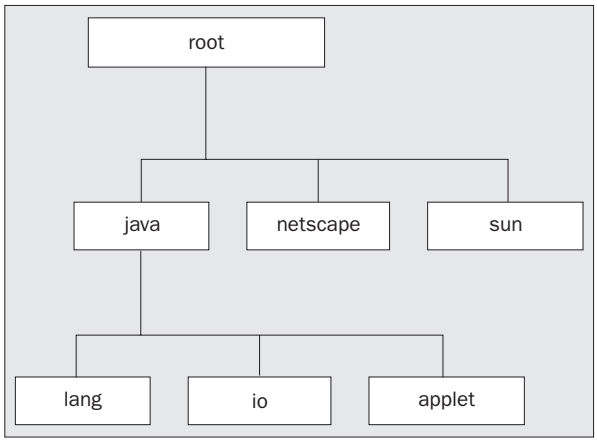
Window.Packages (Property)

A top level `JavaPackage` object that is the root of a tree of Java packages.

Availability:	JavaScript - 1.1 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`JavaPackage object`	
JavaScript syntax:	N	*myWindow*`.Packages`
	N	`Packages`

This property contains a read-only reference to a `JavaPackage` that sits at the top of the Java package hierarchy; the root node of the tree. A `JavaPackage` is a container for other `JavaPackage` objects and `JavaClass` objects. By default, the Netscape Navigator browser will have three packages belonging to this top level node (`java`, `sun` and `Netscape`). Each of these is a package and will contain other packages. There may be additional externally supplied packages over and above these default three items.

MSIE supports different mechanisms for encapsulating Java code.



See also:	`JavaArray` object, `JavaClass` object, `JavaObject` object, `JavaPackage` object, `Packages`, `Window.java`, `Window.netscape`, `Window.sun`

Property attributes:

`ReadOnly`.

Window.pageXOffset (Property)

The amount that a window has been scrolled to the right.

Availability:	JavaScript - 1.2 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myWindow*.pageXOffset
	N	pageXOffset

This read-only integer value tells you how far the window content has been scrolled in the horizontal direction.

Note that Netscape Navigator is somewhat picky about whether you can even scroll the content of a window. It insists on the scrollbars being visible even though the content may not cause them to be active.

Netscape Navigator 4 will let you scroll layers and Netscape 6 provides sufficient DOM standardized CSS positioning controls to satisfy your requirements.

In the MSIE browser, a similar value is available in the `document.body.scrollLeft` property.

See also:	`Document.body`, `Frame` object, `Window` object, `Window.moveBy()`, `Window.moveTo()`, `Window.pageYOffset`, `Window.scrollBy()`

Property attributes:

`ReadOnly`.

Window.pageYOffset (Property)

The amount that a window has been scrolled downwards.

Availability:	JavaScript - 1.2 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	N	*myWindow*.pageYOffset
	N	pageYOffset

This read-only integer value tells you how far the window content has been scrolled in the vertical direction.

Note that Netscape Navigator is somewhat picky about whether you can even scroll the content of a window. It insists on the scrollbars being visible even though the content may not cause them to be active.

Netscape Navigator 4 will let you scroll layers and Netscape 6 provides sufficient DOM standardized CSS positioning controls to satisfy your requirements.

In the MSIE browser, a similar value is available in the `document.body.scrollTop property`.

See also:	`Document.body`, `Frame` object, `Window` object, `Window.moveBy()`, `Window.moveTo()`, `Window.pageXOffset`, `Window.scrollBy()`

Property attributes:

`ReadOnly`.

Window.parent (Property)

A reference to the parent window in a framed pane.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*myWindow*`.parent`
	-	`parent`

For top level windows, the `parent` property is the same as the `self` property.

For framed windows the parent will be a `Window` object.

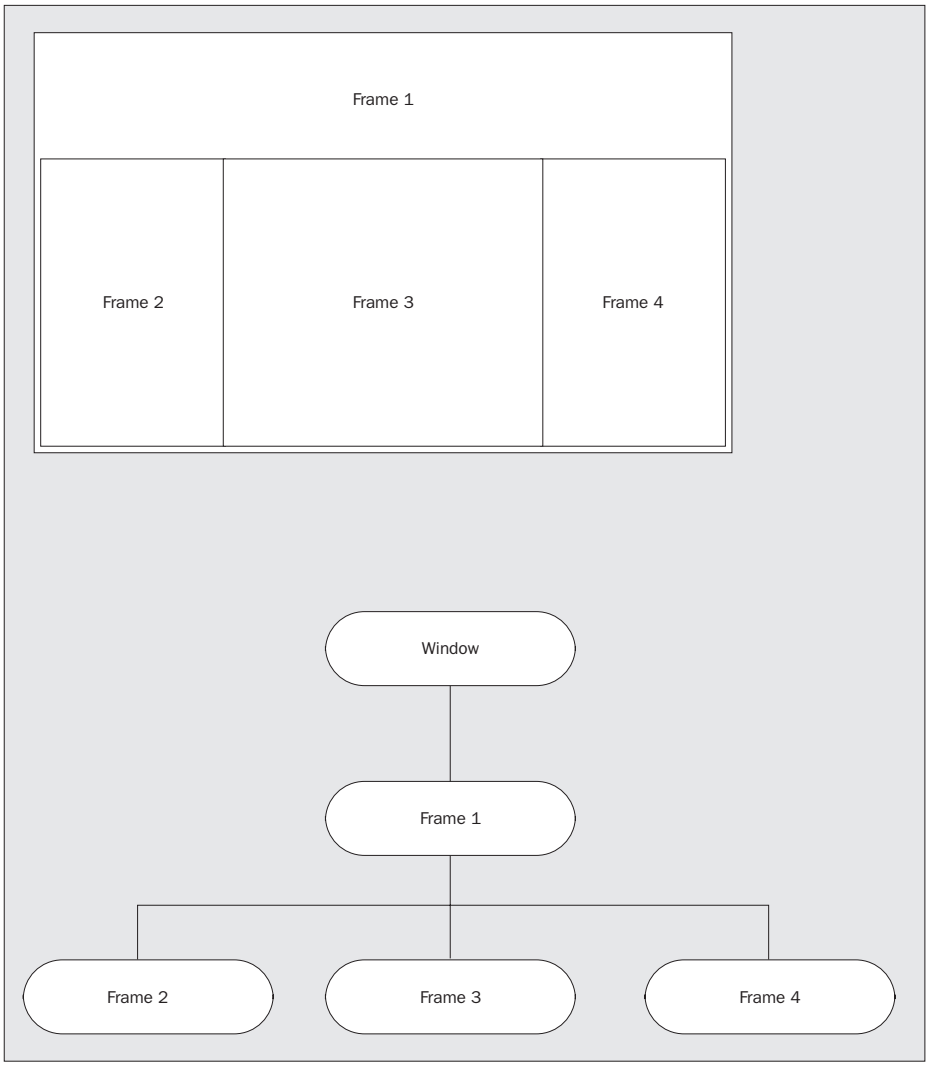
In the same way that you can walk downwards through the frames hierarchy with a `frames[ ... ].frames[ ... ]` construct, a `parent.parent` construct will walk upwards through the hierarchy.

It is important to know where you are so that this is as useful as possible. Relative locations in trees need to have some awareness of the root. If you don't detect this, you might have problems.

Detecting whether you have reached the root can be accomplished by testing whether the parent property of a window refers to the same object as the window itself.

When you reach the top of the tree, the `parent` property should yield the same value as the `self` property. This means that the property will be equivalent to `self` for all windows unless they are inside frames. In MSIE, this also means that they are possibly inside an `<IFRAME>` as well.

Inside a modal dialog window, the parent will be a `Dialog` object.



Warnings:

- Be careful when building tree walking scripts that traverse the window hierarchy. You can reach the top level and continue in an endless loop referring to the parent of an object that is the object itself. You should test to see whether `parent = self` and then exit your tree walker knowing that you have reached the root node.

See also:	`Dialog` object, `Frame` object, `Window` object, `Window.frames[]`, `Window.top`

Property attributes:

`ReadOnly`.

Window.personalbar (Property)

A reference to an object that represents the personal preferences bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`Bar object`	
JavaScript syntax:	-	*myWindow*`.personalbar`
	-	`personalbar`

This is a read-only property containing a reference to a `Bar` object whose visible property contains a Boolean value that controls the visibility of the screen furniture represented by the object.

Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to change the visibility of the personal bar.

Example code:

```
// Request necessary privileges
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
// Hide personal bar
window.personalbar.visible = false;
// There is another way that works without requesting privilege
window.open('', '_top', 'directories=0');
```

See also:	`Bar` object, `Frame` object, `UniversalBrowserWrite`, Window furniture, `Window` object, `Window.locationbar`, `Window.menubar`, `Window.scrollbars`, `Window.statusbar`, `Window.toolbar`

Property attributes:

`ReadOnly`.

Window.pkcs11 (Property)

A hitherto undocumented property of a Netscape Navigator Window object.

Availability:	JavaScript - 1.2 Netscape - 4.04
Property/method value type:	`Pkcs11 object`

JavaScript syntax:	N	*myWindow*.pkcs11
	N	pkcs11

This is part of the Netscape Navigator security model. It's not something you would expect to interact with from JavaScript as a rule.

There are some quite extensive documents covering the use of this facility on the Netscape web site in the developer area.

Some examination with diagnostic scripts does not yield any useful properties, or at least none that can be enumerated. There is a constructor which has a name and prototype property. The prototype of the constructor points back at the `Pkcs11` object itself.

It is somewhat worrying that this object is visible from JavaScript unless there is a genuine reason for being able to access it from script. It may be related to the `crypto` property and also used in conjunction with signed scripts.

If this is a subject you are interested in, there is more information to be studied which is available from a variety of sources. A web search through www.altavista.com yielded over 150 references, some of which document the availability of a Java interface to the PKCS11 security support.

Security in MSIE is supported via other mechanisms and you should search the MSDN web site for relevant links. Links to topics in the MSDN site are relocated from time to time and it is best to search there directly.

See also:	`Pkcs11` object

Property attributes:

`ReadOnly`.

Web-references:

http://developer.netscape.com/support/faqs/pkcs_11.html

Window.print() (Method)

This duplicates the behavior of the Print button on the Netscape Navigator or MSIE button bar.

Availability:	JavaScript - 1.2 JScript - 5.0 Internet Explorer - 5.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*.print()
	-	print()

This is supposed to behave as if the user had clicked on the Print button or selected the Print item from the pull-down menu.

Warnings:

- This does not appear to work on the Macintosh at version 4.7 of Netscape Navigator.

See also:	`Frame` object, `Window` object, `Window.find()`, `Window.home()`,`Window.stop()`

Window.prompt() (Method)

Present a text input prompt box.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*`myResult`* = *`myWindow`*`.prompt(`*`aString`*`,` *`aDefaultValue`*`)`
	-	`myResult = prompt(aString, aDefaultValue)`
Argument list:	*`aDefaultValue`*	An initial content for the text box
	`aString`	Some text to explain what to enter

This presents a modal dialog containing the prompting text message, a text cell to type into and two buttons, OK and Cancel.

This method is useful for debugging where you need to enter values.

Note that the text that is presented in the dialog is unformatted text and you cannot use HTML in the dialog box.

The title bar of the dialog box cannot be changed from its default setting which tells you that the dialog was invoked by JavaScript. In some browsers, it may just display the name of the browser. This is intended to stop script programmers from masquerading their dialog boxes as those of operating system diagnostics and login screens.

The `prompt()` dialog box is modal and blocking. The script must wait for a response from the user.

The value returned by the method is the text typed into the box by the user before pressing the OK button to dismiss the dialog. If the Cancel button is clicked, the method will return `null` instead.

Warnings:

- Earlier documentation refers to the presence of a Clear button but this is not in evidence when testing this function on MSIE and Netscape Navigator. The Clear button is unnecessary since the text is automatically selected and typing into the box clears any previous content. Pressing the Delete key will also clear the text.

See also:	Debugging - client-side, `Dialog` boxes, `Dialog` object, `Frame` object, `Window` object, `Window.alert()`, `Window.confirm()`

Cross-references:

Wrox *Instant JavaScript* – page - 78

Window.releaseEvents() (Function)

Part of the Netscape Navigator 4 event propagation complex.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`myWindow.releaseEvents(anEventMask)`
	N	`releaseEvents(anEventMask)`
Argument list:	*`anEventMask`*	A mask defined with the manifest event constants

This is part of the event management suite which allows events to be routed to handlers other than just the one that defaults to being associated with an event.

The events to be captured are signified by setting bits in a mask.

This method provides a means of indicating which events are no longer needing to be captured by the receiving window object.

The events are specified by using the bitwise OR operator to combine the required event mask constants into a mask that defines the events you want to capture. Refer to the Event Type Constants topic for a list of the event mask values.

We have to implement scripts using this capability if we need to build complex event handling systems on Netcsape 4.x. A different script will be required for MSIE.

You may be able to factor your event handler so that you only have to make platform specific event dispatchers and can call common handling routines that can be shared between MSIE and Netscape.

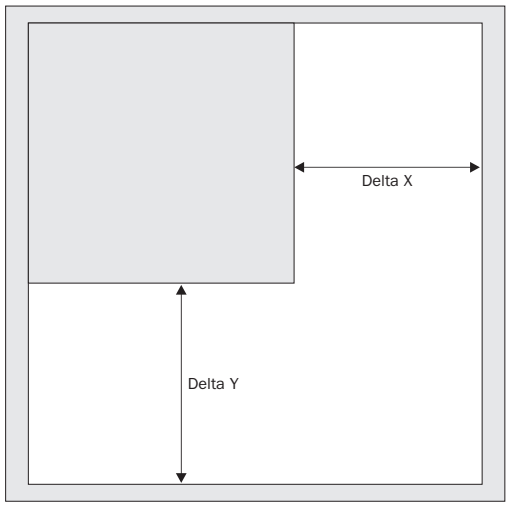
See also:	`captureEvents()`, `Document.captureEvents()`, `Document.releaseEvents()`, `Element.onevent`, Event names, Event propagation, Event type constants, `Event.modifiers`, `Frame` object, `Layer.captureEvents()`, `Layer.releaseEvents()`, `onMouseMove`, `Window` object, `Window.captureEvents()`

Window.resizeBy() (Method)

Resize the window by a specified amount.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*`.resizeBy(`*aChangeX*`, `*aChangeY*`)`
	-	`resizeBy(`*aChangeX*`, `*aChangeY*`)`
Argument list:	*aChangeX*	A difference in pixels
	aChangeY	A difference in pixels

The window size will be altered by the horizontal and vertical amounts specified in the two parameters. The left and top edges will stay in the same place but the right and bottom edges will be adjusted to the new positions.



Warnings:

- There are some security implications for resizing windows in Netscape 4. This is to prevent you hiding a window by making it too small.
- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to set the window size to less than 100x100 pixels.

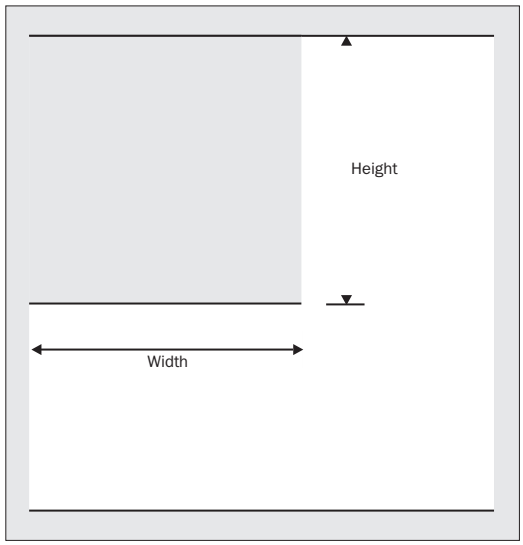
See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.moveBy()`, `Window.moveTo()`, `Window.onresize`, `Window.resizeTo()`

Window.resizeTo() (Method)

Resize the window to specified dimensions.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`myWindow.resizeTo(aSizeX, aSizeY)`
	-	`resizeTo(aSizeX, aSizeY)`
Argument list:	*`aSizeX`*	A distance in pixels
	`aSizeY`	A distance in pixels

The window size will be altered to the horizontal and vertical values specified in the two parameters. The left and top edges will stay in the same place but the right and bottom edges will be adjusted to reflect the new width and height.



Warnings:

- There are some security implications for resizing windows in Netscape 4. This is to prevent you hiding a window by making it too small.
- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to set the window size to less than 100x100 pixels.

See also:	`Frame` object, `UniversalBrowserWrite`, `Window` object, `Window.moveBy()`, `Window.moveTo()`, `Window.onresize`, `Window.resizeBy()`

Window.returnValue (Property)

The return value for a modal dialog window.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`User defined`	
JavaScript syntax:	IE	`returnValue`
Argument list:	*`aNewValue`*	The value to be returned when the modal dialog closes

The MSIE browser supports a special kind of window called a Modal Dialog. This is similar to using the `open()` method to create a new window except that it is forced to always be on the top and you cannot click outside of it to bring focus to another window within the browser.

When the modal window exits, you have the opportunity to store a value in the `returnValue` property. Since the window is instantiated by a method call, the caller will expect a value to be returned. This value will be null unless you specify something yourself.

Since you can pass arguments into the modal window, this provides a way to transmit user interaction dependent values back to the caller.

Warnings:

- How this behaves in a truly multi-process environment such as Windows NT or Mac OS X may be slightly different. In an operating system such as those, modality may only extend within the application and it may be possible to switch to a different application. However on returning to the MSIE browser, the modality will still be in force.

See also:	`Frame` object, `Window` object, `Window.showModalDialog()`

Window.routeEvent() (Function)

Part of the Netscape Navigator 4 event propagation complex.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	`myWindow.routeEvent(`*`anEvent`*`)`
	N	`routeEvent(`*`anEvent`*`)`
Argument list:	*`anEvent`*	An event object

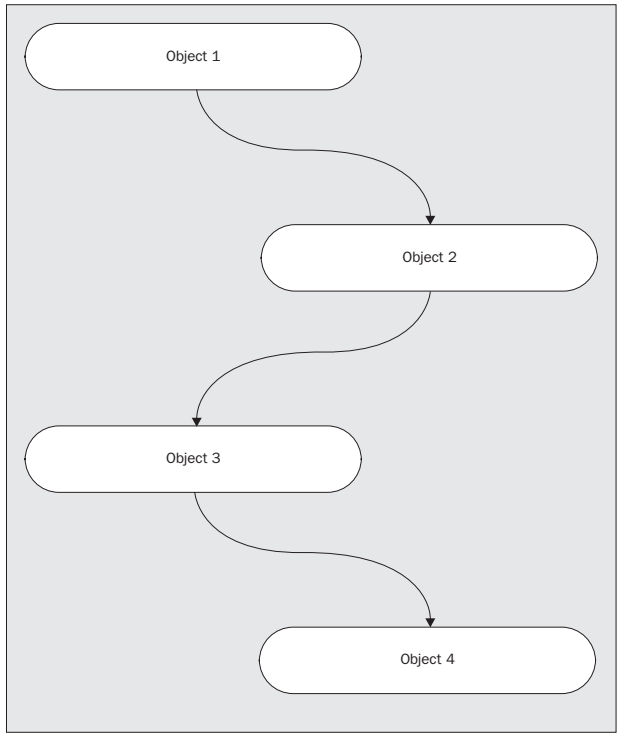
This is part of the event management suite which allows events to be routed to handlers other than just the one that defaults to being associated with an event.

Depending on the current setting of the event type mask for the receiving object, an event may be available for processing with this method. An initial mask that enumerates the events to be received might have been set up with the `captureEvents()` method. Then perhaps sometime later, some of those events may have been deselected by means of the `releaseEvents()` method. The remaining mask defines the finite set of events that will be visible to the object.

So, when a suitable event arrives and is captured, it can be passed on to the appropriately mapped event handler function belonging to the next object in the event handling hierarchy or to another handler belonging to the receiving object.

This means that an event can be processed via several handlers within an object before being passed to the next object in sequence. The order in which handlers and objects are visited by the event is controlled by Netscape Navigator. However, whether the event is passed on or not is controlled by the script in the handler. If it chooses not to pass on the event, then no subsequent handlers will see it, nor will any other objects.

This trickle-down effect is somewhat clumsy and uncontrolled. You can gain some control over the way that events are propagated with the `handleEvent()` method and applying it to the various objects that need to take the event. This does disperse the event handling around the scripts somewhat and can be difficult to maintain.



Warnings:

- This entire mechanism may become obsolete when the W3C standardizes the event handling process for the Document Object Model.

See also:	`captureEvents()`, Event handler, Event management, Event propagation, `Frame` object, `handleEvent()`, `Layer.routeEvent()`, `Window` object, `Window.handleEvent()`

Window.screen (Property)

A reference to a `Screen` object that the window is being displayed in.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0 Opera - 5.0	
Property/method value type:	`Screen object`	
JavaScript syntax:	-	`myWindow.screen`
	-	`screen`

The object returned by this property is shared by all windows since there is only one `Screen` object for the entire browser.

Refer to the `Screen` object topic for details of how you can examine its properties. There are some minor differences between MSIE and Netscape Navigator regarding the properties that their `Screen` objects support.

The WebTV box yields an object that describes the properties of a device that operates at TV display resolution.

Warnings:

- This may not behave the same on all platform configurations. Consider the possibility of having several monitor screens. Some OS platforms insist on all screens being the same size and resolution in which case a single `Screen` object will suffice.
- In the case of the Macintosh Operating System, you may have several screens, each of which can be a different size, orientation and color depth. Indeed, you can even mix monochrome and color screens. One will be the main screen by virtue of it having the menubar placed on it. The screens can be positioned relative to one another in any orientation.
- So for an OS such as Mac OS, a single `Screen` object may not suffice. However, the browser manufacturers may not fully support this. You will need to test this if it is important to your application. It is possible there will still be a single `Screen` object and that it will represent the master screen. However it might represent the screen that the window is positioned on.

- The Macintosh OS is somewhat unique in that a window may span several screen displays. This would make it difficult for any `Screen` object (or even several) to accurately represent the real world.

See also:	JellyScript, `Screen` object

Property attributes:

`ReadOnly`.

Window.screenLeft (Property)

The left edge of the screen.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myWindow*`.screenLeft`
	IE	`screenLeft`

This is the offset of the left edge of the window from the edge of the screen display.

The equivalent value in Netscape Navigator can be obtained from the `Window.screenX` property.

See also:	`Window.screenX`

Property attributes:

`ReadOnly`.

Window.screenTop (Property)

The top edge of the screen.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myWindow*`.screenTop`
	IE	`screenTop`

This is the offset of the top edge of the window from the top of the screen display.

The equivalent value in Netscape Navigator can be obtained from the `Window.screenY` property.

See also:	Window.screenY

Property attributes:

ReadOnly.

Window.screenX (Property)

The X coordinate of the window within the screen display.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	Number primitive	
JavaScript syntax:	N	*myWindow*.screenX
	N	*myWindow*.screenX = *aCoordinate*
	N	screenX
	N	*screenX* = *aCoordinate*
Argument list:	*aCoordinate*	A pixel position on the screen

This is the horizontal coordinate of the left edge of the Netscape Navigator window. For a Window object that represents the contents of a frame in a frame-set, this value will be the left edge of the containing top level window.

The equivalent value in MSIE can be obtained from the Window.screenLeft property.

See also:	Window.screenLeft

Window.screenY (Property)

The Y coordinate of the window within the screen display.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	Number primitive	
JavaScript syntax:	N	*myWindow*.screenY
	N	*myWindow*.screenY = *aCoordinate*
	N	screenY
	N	screenY = *aCoordinate*
Argument list:	*aCoordinate*	A pixel position on the screen

This is thevertical coordinate of the top edge of the Netscape Navigator window. For a `Window` object that represents the contents of a frame in a frame-set, this value will be the top edge of the containing top level window.

The equivalent value in MSIE can be obtained from the `Window.screenTop` property.

See also:	`Window.screenTop`

Window.scroll() (Method)

This is equivalent to the `scrollTo()` method but has been retained for backwards compatibility.

Availability:	JavaScript - 1.1 JScript - 3.0 Internet Explorer - 4.0 Netscape - 3.0 Opera - 3.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	`myWindow.scroll(aPositionX, aPositionY)`
	-	`scroll(aPositionX, aPositionY)`
Argument list:	*`aPositionX`*	A position in pixels
	`aPositionY`	A position in pixels
Deprecated:	JavaScript - 1.2	

The window content will be scrolled by the amount specified in the X and Y values.

Warnings:

- You should use the `scrollTo()` method in preference to this one as it is now deprecated.
- Vertical scrolling in Netscape Navigator moves in the reverse direction to MSIE. Because the standard is ambiguous on this point, you will need to check this on other implementations.
- Netscape Navigator will only permit a document to be scrolled if the scroll bars are active and visible. MSIE does not care whether the scroll bars are visible or whether you are scrolling the document past its end point. Netscape Navigator however allows layers to be scrolled anyhow.

See also:	`Frame` object, `Window` object, `Window.scrollBy()`, `Window.scrollTo()`

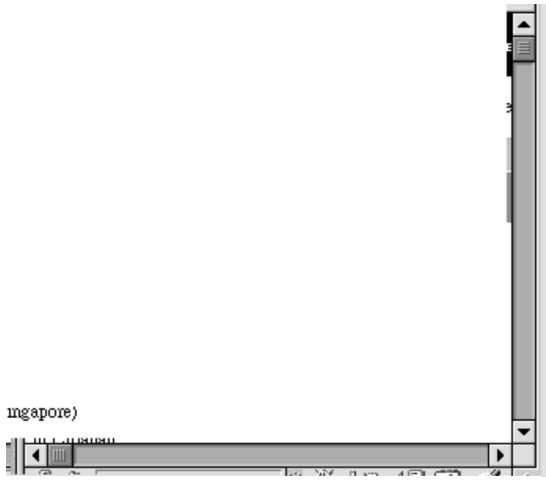
Window.scrollbars (Property)

A reference to an object that represents the scroll bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`Bar object`	
JavaScript syntax:	-	*myWindow*.scrollbars
	-	scrollbars

This is a read-only property containing a reference to a `Bar` object whose visible property contains a Boolean value that controls the visibility of the screen furniture represented by the object.

In this case, it is supposed to control the visibility of the scroll bars.



Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to change the visibility of the scrollbars.

Example code:

```
// Request necessary privileges
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
// Hide scroll bars
window.scrollbars.visible = false;
// There is another way that works without requesting privilege
window.open('', '_top', 'scrollbars=0');
```

See also:	`Bar` object, `Frame` object, `UniversalBrowserWrite`, Window furniture, `Window` object, `Window.locationbar`, `Window.menubar`, `Window.personalbar`, `Window.statusbar`, `Window.toolbar`

Property attributes:

`ReadOnly`.

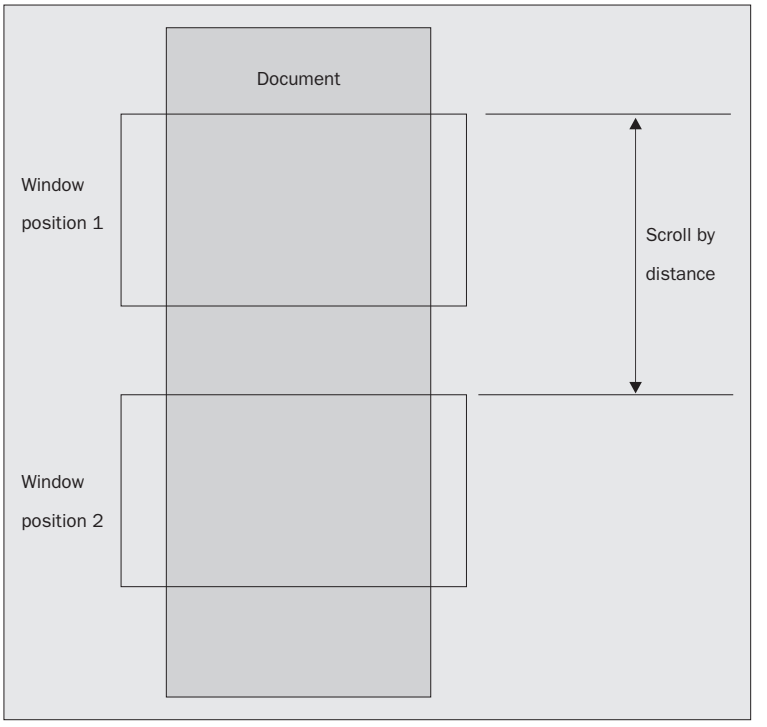
Window.scrollBy() (Method)

Scroll the document in the window by a specific amount.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*`.scrollBy(`*anOffsetX*`,` *anOffsetY*`)`
	-	`scrollBy(`*anOffsetX*`,` *anOffsetY*`)`
Argument list:	*anOffsetX*	A distance in pixels
	anOffsetY	A distance in pixels

This method will scroll the window relative to its current position by the indicated amount. You can scroll in either the horizontal or vertical axis or even both at once.

Although the method is supported by MSIE and Netscape Navigator, they scroll in opposite directions vertically. It was probably too much to expect them to support the feature in the same way.



Warnings:

- Vertical scrolling in Netscape Navigator moves in the reverse direction to MSIE. Because the standard is ambiguous on this point, you will need to check this on other implementations.
- Netscape Navigator will only permit a document to be scrolled if the scroll bars are active and visible. MSIE does not care whether the scroll bars are visible or whether you are scrolling the document past its end point. Netscape Navigator however allows layers to be scrolled anyhow.

See also:	`Frame` object, `Window` object, `Window.moveBy()`, `Window.moveTo()`, `Window.pageXOffset`, `Window.pageYOffset`, `Window.scroll()`, `Window.scrollTo()`

Window.scrollTo() (Method)

Scroll the document in the window to a specific location.

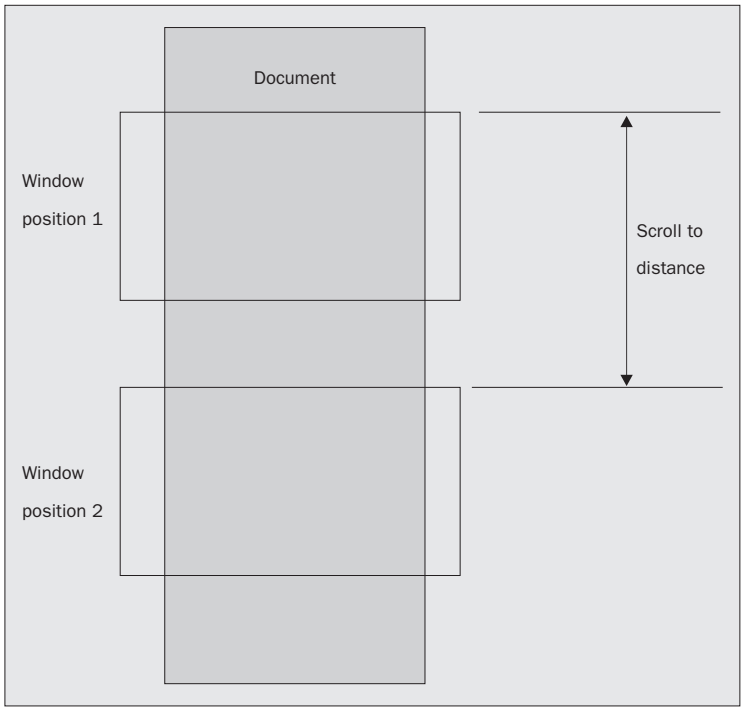
Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*`.scrollTo(`*aPositionX*`,` *aPositionY*`)`
	-	`scrollTo(`*aPositionX*`,` *aPositionY*`)`
Argument list:	*aPositionX*	A location in pixels
	aPositionY	A location in pixels

This method will scroll the window to an absolute position specified by the two values. You can scroll in either the horizontal or vertical axis or even both at once.

Although the method is supported by MSIE and Netscape Navigator, they scroll in opposite directions vertically.

You can use this scrolling facility to bring a specific anchor into view. You would do this by enquiring of the anchor what its X and Y coordinates are and then using that as a scroll destination.

You should use this method in preference to the `window.scroll()` method which is now deprecated.



Warnings:

- Vertical scrolling in Netscape Navigator moves in the reverse direction to MSIE. Because the standard is ambiguous on this point, you will need to check this on other implementations.
- Netscape Navigator will only permit a document to be scrolled if the scroll bars are active and visible. MSIE does not care whether the scroll bars are visible or whether you are scrolling the document past its end point. Netscape Navigator however allows layers to be scrolled anyhow.

See also:	`Anchor` object, `Frame` object, `Window` object, `Window.scroll()`, `Window.scrollBy()`

Window.secure (Property)

A flag indicating that a window was loaded from a secure source.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	N	*myWindow*`.secure`
	N	`secure`
Argument list:	*false*	Window is not currently securely served
	true	Window was loaded from a secure location

This value will be set to `true` if the window contents were delivered from a secure source (probably via the `https:` protocol). Normally this value will be `false`, as is the case if the window is loaded from a local file. That would seem a little strange in that you probably have to have local access control permissions to reach a file on the local file system and one would have thought that would be secure and meriting a `true` value in this property.

Property attributes:

`ReadOnly`.

Window.self (Property)

A reference to the window itself.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*myWindow*`.self`
	-	`self`

This is another name for the `window.window` property in this context. However, `self` is useful because you can build reusable scripts with it that can be used with a variety of object types and instances. Don't forget that this can also refer to a frame as well as a window since they are both represented by the `window` object.

The `self` property can be used without the `window` prefix because it belongs to the global object in a web browser window. Using the `self` keyword like this makes no difference to the functionality of your script but it does make it easier to understand. For the same reason, you may want to use the `window` property in the same way.

See also:	Frame object, `self`, `Window` object, `Window.frame`, `Window.window`

Property attributes:

`ReadOnly`.

Window.setHotkeys() (Method)

Activate or deactivate keyboard shortcuts for this window.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myWindow*`.setHotKeys(`*aSwitch*`)`
	N	`setHotKeys(`*aSwitch*`)`
Argument list:	*aSwitch*	A Boolean switch value

This provides a way to turn keyboard short-cuts on and off. For windows that you create with the `window.open()` method, you can control this with the feature list that you pass to the `open()` method.

Pass a `true` value to activate the hot keys and a `false` value to deactivate them.

Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to call this method in Netscape Navigator.
- Beware of the spelling, the word 'keys' is not capitalized inside this function name.

See also:	`UniversalBrowserWrite`

Window.setInterval() (Method)

Schedule a function to be executed at regular intervals.

Availability:	JavaScript - 1.2 JScript - 3.0 Internet Explorer - 4.0 Netscape - 4.0
Property/method value type:	`Number primitive`

JavaScript syntax:	-	*myWindow*.setInterval(*aFunction*, *anInterval*)
	-	*myWindow*.setInterval(*aFunction*, *anInterval*, *someArguments*)
	-	myWindow.setInterval(*aFunction*, *anInterval*, *someArguments*, *aLanguage*)
	-	myWindow.setInterval(*aSourceText*, *anInterval*)
	-	myWindow.setInterval(*aSourceText*, *anInterval*, *someArguments*)
	-	*myWindow*.setInterval(*aSourceText*, *anInterval*, *someArguments*, *aLanguage*)
	-	setInterval(*aFunction*, *anInterval*)
	-	setInterval(*aFunction*, *anInterval*, *someArguments*)
	-	setInterval(*aFunction*, *anInterval*, *someArguments*, *aLanguage*)
	-	setInterval(*aSourceText*, *anInterval*)
	-	setInterval(*aSourceText*, *anInterval*, *someArguments*)
	-	setInterval(*aSourceText*, *anInterval*, *someArguments*, *aLanguage*)
Argument list:	*aFunction*	A function object
	aLanguage	A scripting language to execute the script source (MSIE only)
	anInterval	A time interval in milliseconds
	aSourceText	Some valid script source text
	someArguments	The arguments to the function object (not supported in MSIE version 4)

The `setInterval()` method establishes a periodically scheduled execution timer that runs the same fragment of script continuously with a delay timer between each cycle.

If you only want to delay the execution of some code and you want it to be executed just once, then use the `setTimeout()` method instead. That will defer execution and clear its timer automatically when it executed.

Passing a function as one of the arguments is only supported in MSIE from version 5.0 upwards and Netscape Navigator from version 4 upwards. You can only specify the scripting language in MSIE however.

The simpler form in which a script source text can be passed in a string with a second argument to specify the interval is much more portable and recommended for use. You can pass a multiple line script fragment in this argument as long as each line is separated by a semi-colon. Functionally, this is very similar to the `eval()` method with an extra repeat periodicity value.

Be careful that you note the value returned by this method if you intend to deactivate the periodic execution. Without that value, you have no way to identify which one of possibly several timers you want to cancel, and take care not to cancel the timer more than once. Cancelling a non-existent periodic timer or deferred action is likely to crash your browser.

The result of this method is an identifying value that can be used with the `clearInterval()` method to cancel this periodic execution.

Warnings:

- The functionality of this method is more limited in MSIE version 4. It does not support the passing of a function object and its arguments in separate parameters. However since you can define a fragment of JavaScript to call a function there are few circumstances where this will be a problem. It does prevent you from manufacturing a function object and calling it though.
- On the other hand, implementing something that is non-portable across the MSIE and Netscape Navigator browsers is a bad idea anyway.
- Be careful how you operate on these interval timers. You can store an identifier, which you can later use to cancel the timer. However, if the timer has already been fired, some browsers may crash due to you trying to cancel a non-existent timer.
- Write your periodically executed code carefully to avoid memory leaks, as any repetitive code that leaks memory will rapidly slow the performance of the user's browser. It is not hard to leak as much as 50K bytes per loop in JavaScript. This will rapidly fill your memory and eventually the browser may crash. Even worse, memory leaks have a nasty habit of bringing the operating system down as well.

See also:	`eval()`, `Frame` object, Interval handlers, Memory leak, Timeout handlers, `Window` object, `Window.clearInterval()`, `Window.setTimeout()`

Window.setResizable() (Method)

Enable or inhibit the window resize capability.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myWindow*`.setResizable(`*aSwitch*`)`
	N	`setResizable(`*aSwitch*`)`
Argument list:	*aSwitch*	A Boolean value to control the functionality

This provides a way to allow or deny user access to window resizing facilities.

This switches the resize feature on and off, which is something you can control in new windows you create with the `open()` method when you specify the resizable feature.

Your script needs `UniversalBrowserWrite` privileges to use this method.

Warnings:

- Some platforms will not support this facility due to the way their window manager operates. This is likely to be supported on desktop systems and less likely to work on workstations that use X-Windows.

See also:	`UniversalBrowserWrite`

Window.setTimeout() (Method)

A timeout control method.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	-	*myWindow*.setTimeout(*aFunction*, *aWaitTime*)
	-	*myWindow*.setTimeout(*aFunction*, *aWaitTime*, *someArguments*)
	-	*myWindow*.setTimeout(*aFunction*, *aWaitTime*, *someArguments*, *aLanguage*)
	-	*myWindow*.setTimeout(*aSourceText*, *aWaitTime*)
	-	*myWindow*.setTimeout(*aSourceText*, *aWaitTime*, *someArguments*)
	-	*myWindow*.setTimeout(*aSourceText*, *aWaitTime*, *someArguments*, *aLanguage*)
	-	setTimeout(*aFunction*, *aWaitTime*)
	-	setTimeout(*aFunction*, *aWaitTime*, *someArguments*)
	-	setTimeout(*aFunction*, *aWaitTime*, *someArguments*, *aLanguage*)
	-	setTimeout(*aSourceText*, *aWaitTime*)
	-	setTimeout(*aSourceText*, *aWaitTime*, *someArguments*)
	-	setTimeout(*aSourceText*, *aWaitTime*, *someArguments*, *aLanguage*)
Argument list:	*aFunction*	A function object
	aLanguage	A scripting language to execute the script source (MSIE only)
	aWaitTime	A time interval in milliseconds
	aSourceText	Some valid script source text
	someArguments	The arguments to the function object (not supported in MSIE version 4)

The `setTimeout()` method provides a way to defer the execution of a fragment of script source text. It is analogous to the `eval()` method with a delay before execution. Although this method returns an ID value that can then be used with the `clearTimeout()` method to cancel the execution, it is very likely that the deferred code will have executed already. You should set a flag accordingly so that you can avoid killing a deferred task that has already been completed. Attempting to kill deferred tasks that are no longer pending can crash your browser.

Code executed by this deferred mechanism will only be executed once. If you want it to be executed continuously then `setInterval()` is a better alternative. On the other hand, you may want to conditionally defer it again in which case you should call a handler function and, before exiting it, make another call to `setTimeout()` to activate another deferred task.

As is the case with `setInterval()` and `eval()`, you can execute multiple statements as long as they are separated by semi-colons.

Passing a function as one of the arguments is supported by MSIE in version 5 upwards and by Netscape Navigator in version 4 upwards. The scripting language can only be defined in MSIE.

This facility may be used to present a message in the status bar and then clear it again after some period of time has elapsed. It can be used to generate some animation in the status bar, although many people consider this to be a design cliche and much overused. If you do animate the status bar, you should consider whether it is useful and not distracting.

These timed animations are generally best triggered by an `onLoad` event handler.

Note that the deferred code is executed in the context and scope chain of the `window` object that received the method call to set up the deferred task.

The example is a cut down version of the ticker script used in the BBC News Online web site. The display techniques are the same but the example only shows one story. To see the real ticker in operation, refer to http://www.bbc.co.uk/news and view it with an MSIE browser. In the News Online ticker, many coding compromises were necessary to work round object boundary bugs in the MSIE for Macintosh browser. Because the ticker is constantly being updated, the object boundary is changing all the time and although this was played in an `<IFRAME>`, the mouse enter and mouse out events caused the MSIE browser to crash frequently. Earlier versions of the ticker also did a large amount of string creation/destruction which caused somewhat massive memory leaks. You can alleviate this by using meta refresh tags to force the garbage collection to happen.

Warnings:

- Be careful how you operate on these interval timers. You can store an identifier, which you can later use to cancel the timer. However, if the timer has already been fired, some browsers may crash due to you trying to cancel a non-existent timer.
- This method leaks somewhat badly in Netscape 2.

Example code:

```
<HTML>
<HEAD>
<STYLE  TYPE="text/css">
<!--
A
{
   font-family: Verdana, Arial, Helvetica, sans-serif, "MS sans serif";
   font-size: 11px;
   line-height: 11px;
   text-decoration: none;
   color: #333366;
   font-weight: bold;
}

A.latest
{
   color: #CC3300;
}

A:hover
{
   color: #CC3300;
}
-->
</STYLE>
<SCRIPT LANGUAGE=JAVASCRIPT>
// This script will only work on MSIE
var theTickerText       = "The ticker text goes here and plays out until it is
finished before repeating again.";
var theCharacterTimeout = 45;
var theStoryTimeout     = 5000;
var theEnumerator       = 0;
// --- Run the ticker
function doTheTicker()
{
   if((theEnumerator % 2) == 1)
   {
      writeTicker("_");
   }
   else
   {
      writeTicker("*");
   }

   theEnumerator++;

   if(theEnumerator == theTickerText.length+1)
   {
      writeTicker("");
      theEnumerator = 0;
      setTimeout("doTheTicker()", theStoryTimeout);
   }
   else
   {
      setTimeout("doTheTicker()", theCharacterTimeout);
   }
}
```

```
function writeTicker(aWidget)
{
   document.all.hottext.innerHTML = theTickerText.substring(0,theEnumerator) +
aWidget;
}
</SCRIPT>
</HEAD>
<BODY onLoad="doTheTicker();">
<A ID="latest" CLASS="latest" HREF="/" target=_top>LATEST: </A><A ID="hottext"
HREF="/"></A>
</BODY>
</HTML>
```

See also:	`clearTimeout()`, `eval()`, `Frame` object, Memory leak, Timeout handlers, `Window` object, `Window.clearInterval()`, `Window.clearTimeout()`, `Window.setInterval()`

Window.setZOptions() (Method)

Define the window stacking behavior.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	N	*myWindow*`.setZOptions(`*anOptionValue*`)`
	N	`setZOptions(`*anOptionValue*`)`
Argument list:	*anOptionValue*	One of a range of possible settings for the feature

When you call the `window.open()` method, there are several feature values that can be used to control the Z-Ordering of the windows. This method provides a way to modify the settings of the Z-Ordering after the window has already been created.

The option value should be one of the following:

Feature	Description
`alwaysRaised`	The window should always be at the top of the stack of windows even when it does not have input focus.
`alwaysLowered`	The window should always be at the bottom of the stack of windows even when it does have the input focus.
`z-lock`	The window should always be at the same Z position in the stack of windows regardless of which window has the focus.
empty	The window exhibits normal stacking behavior; it is brought to the top when it has the focus.

These options are spelled the same as they are when you use them in the `window.open()` method. Naturally, they are mutually exclusive and you can use only one of them at a time.

With this special Netscape Navigator functionality and some careful use of the `window.open()` method, you can effectively simulate the same behavior as that of the MSIE supported modal window. You will need to use global variables and intra-window method calls to pass arguments, however.

Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to call this method in Netscape Navigator.

See also:	`UniversalBrowserWrite`, `Window.showModalDialog()`

Window.showHelp() (Method)

Display the help window.

Availability:	JScript - 3.0 Internet Explorer - 4.0	
Property/method value type:	`Window object`	
JavaScript syntax:	IE	`myWindow.showHelp(`*aURL*`)`
	IE	`showHelp(`*aURL*`)`
Argument list:	*aURL*	The URL of a help page

This will call up the online help facility of the browser.

Warnings:

- This is not supported on the Macintosh platform.

See also:	Dialog boxes, `Dialog` object

Window.showModalDialog() (Method)

Display a modal dialog.

Availability:	JScript - 3.0 Internet Explorer - 4.0
Property/method value type:	`User defined`

JavaScript syntax:	IE	*myWindow*.showModalDialog(*aURL*, *someArguments*)
	IE	*myWindow*.showModalDialog(*aURL*, *someArguments*, *someFeatures*)
	IE	showModalDialog(*aURL*, *someArguments*)
	IE	showModalDialog(aURL, *someArguments*, *someFeatures*)
Argument list:	*aURL*	A URL to load into the modal dialog
	someArguments	Aggregated arguments to pass to the modal dialog
	someFeatures	Window adornment features

The MSIE browser supports a special window that behaves like a normal window, except that it does not permit any other action until it is dismissed.

To present a modal dialog, you call this method which does not return until the window is dismissed. When the window is dismissed, whatever the `returnValue` property of that window was set to will be returned as the result of this method call.

You must specify a URL to be loaded into the window and you can also pass arguments to it to control the behavior of any content with some scripts that get loaded with the page. It is that script code which will ultimately store something in the `returnValue` property of the window before calling `window.close()` to dismiss the window. The arguments are passed by aggregating them into an array or object which is then 'unpacked' inside the modal dialog's context.

This is only supported on MSIE but you can simulate most of its functionality quite easily in Netscape Navigator. Passing values in and out becomes somewhat tricky but you can accomplish that by passing property values via calls to scripts in other windows or by setting properties belonging to objects that you know will persist long enough for them to be retrieved. However, note that you may not be able to simulate the interlocking modality quite as easily.

A third optional argument can be specified in a similar way to the feature list of a `window.open()` method. You might put values such as this in the third argument to control the size of the new dialog:

```
"dialogWidth:5cm; dialogHeight:10cm; dialogTop:0cm; dialogLeft:0cm"
```

Note that they are a number and a measurement unit in the manner of the CSS positioning style controls. If you don't specify a measurement unit, then MSIE 4 assumes you are measuring in ems and MSIE 5 assumes pixels.

Here is a list of the features that can be applied to a modal dialog window as it is opened:

Feature	Range	Default	Description
`center:`	`yes`, `no`, `1`, `0`, `on`, `off`	`yes`	Controls dialog window centering within the desktop.
`dialogHeight:`	height value	none	Sets the dialog window height
`dialogHide:`	`yes`, `no`, `1`, `0`, `on`, `off`	`no`	Controls the dialog window visibility when printing or using print preview. This feature is only available when a dialog box is opened from a trusted application.
`dialogLeft:`	left position	none	Sets the left edge coordinate of the dialog window relative to the upper-left corner of the desktop.
`dialogTop:`	top position	none	Sets the top edge coordinate of the dialog window relative to the upper-left corner of the desktop.
`dialogWidth:`	width value	none	Sets the dialog window width
`edge:`	`sunken`, `raised`	`raised`	Defines the dialog window edge style
`help:`	`yes`, `no`, `1`, `0`, `on`, `off`	`yes`	Controls the context-sensitive Help icon.
`resizable:`	`yes`, `no`, `1`, `0`, `on`, `off`	`no`	Controls the resize box.
`scroll:`	`yes`, `no`, `1`, `0`, `on`, `off`	`yes`	Defines whether the dialog window has scrollbars.
`status:`	`yes`, `no`, `1`, `0`, `on`, `off`	Varies	Defines whether the dialog window has a status bar. The default is `yes` for dialog windows that aren't trusted and `no` for trusted dialog windows.
`unadorned:`	`yes`, `no`, `1`, `0`, `on`, `off`	`no`	Controls the border window chrome. This feature is only available when a dialog box is opened from a trusted application.

The result of this method call is the value stored in the `returnValue` property of the modal window before it is dismissed.

The example demonstrates how to pass some argument values in an array and how to return a value entered by the user.

Warnings:

- Note that the feature list for a modal dialog is different to the feature list used in a `window.open()` method.
- The Macintosh version of MSIE does not appear to support the dialog sizing controls although you can write and then read back values in the properties that control size and position.
- On the Macintosh variant of MSIE version 5, a modal dialog is displayed but if its contents are scrolled, they can degenerate the off-screen buffer resulting in weird tearing effects in the image data. This may be due to having an <IFRAME> in the window or it may be caused by a timing problem with multiple clicks on the scroll bar in rapid succession.

Example code:

```
<!-- Save this in demo.html -->
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT>
myArray = new Array(100, 200, "XXX", "YYY");
myReturnValue = showModalDialog("./page.html", myArray);
alert("Value returned from window is "+myReturnValue);
</SCRIPT>
</BODY>
</HTML>
-----------------------------------------------------------------
<!-- Save this in page.html in the same folder -->
<HTML>
<HEAD>
</HEAD>
<BODY>
Hi there!!<HR>Input values are:
<BR>
<BR>
<SCRIPT>
for(myEnum=0; myEnum<dialogArguments.length; myEnum++){   document.write(myEnum);
document.write(" - ");   document.write(dialogArguments[myEnum]);
document.write("<BR>");
}
returnValue = prompt("What do you want to send back");
</SCRIPT>
</BODY>
</HTML>
```

See also:	Dialog boxes, `Dialog` object, `Frame` object, `Window` object, `Window.dialogArguments`, `Window.returnValue`, `Window.setZOptions()`

Window.showModelessDialog() (Method)

Display a modeless dialog window.

Availability:	JScript - 5.0 Internet Explorer - 5.0	
Property/method value type:	`Window object`	
JavaScript syntax:	IE	*myWindow*.showModelessDialog(*aURL*, *someArguments*)
	IE	*myWindow*.showModelessDialog(*aURL*, *someArguments*, *someFeatures*)
	IE	showModelessDialog(*aURL*, *someArguments*)
	IE	showModelessDialog(*aURL*, *someArguments*, *someFeatures*)

Argument list:	*aURL*	A URL to load into the modeless dialog
	someArguments	Arguments to pass to the modeless dialog
	someFeatures	Window adornment features

The MSIE browser supports a special window that behaves like a modal dialog but can be switched into the background. This is therefore a modeless dialog. Because it does not operate in a modal fashion, it does not lock out other activity in the browser. Having called this method to display a modeless dialog, the function returns immediately and the script that called it will continue execution. The opening window can continue to receive input if necessary, which you might find useful for opening palette windows, for example.

The result of this method call is a reference to the window object that represents the modeless dialog.

You must specify a URL value in the first argument for a document to be loaded into the window.

The second argument passes some arguments to the modeless dialog in the same way as you might to a modal dialog.

A third optional argument can be specified in a similar way to the feature list of a `window.showModalDialog()` method. However, note that the feature list is different to that for a normal window created with a `window.open()` method call. Refer to the `window.showModalDialog()` topic for details of the feature list for modal or modeless dialog windows.

See also:	Dialog boxes, `Dialog` object`Window.showModalDialog()`

Window.sidebar (Property)

A reference to an object that represents the sidebar frame in Netscape 6.0.

Availability:	JavaScript - 1.5 Netscape - 6.0	
Property/method value type:	`Sidebar object`	
JavaScript syntax:	N	`myWindow.sidebar`
	N	`sidebar`

This is a new property introduced to support the sidebar in Netscape 6. In essence this introduces a permanent second frame that can be used to hold navigational content. Actually its capabilities seem quite powerful and since it can have HTML loaded into it and can be addressed with JavaScript some interesting possibilities await us as we explore it further.

See also:	`Sidebar` object

Window.status (Property)

A property containing the text displayed in the status bar.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	-	*myWindow*.status
	-	*myWindow*.status = aString
	-	status
	-	status = *aString*
Argument list:	*aString*	A string to display in the status bar

This property should be used to set a transient value for the status bar when providing some rollover management.

This is ideal for use in event handlers or fragments of code being executed with a deferred task. Normally setting this from a script would display the status text only momentarily until the user moved the mouse or the document finished loading.

When the status text is erased, the defaultStatus value of the window will usually be the text that overwrites it, unless some other text is called in the meantime. It really depends on what event handlers are invoked as the mouse moves.

In Netscape Navigator, setting the status value in a submit button `onClick` handler instantly updates the value displayed in the status bar. However as soon as you roll off the button, the `defaultStatus` value is displayed instead and the transient status value is lost. Rolling back onto the submit button won't display it again because it was set by the `onClick` handler.

In MSIE, the behavior of the status and `defaultStatus` properties is so nearly identical as to be hard to distinguish one from the other.

The status message box belongs to the top level window. This means that it behaves with some slight differences if there are multiple frames and the status and `defaultStatus` properties are set from inside them. This is also prone to differences between browsers and it is recommended that you experiment somewhat to achieve exactly the desired behavior you want to make sure it is consistent across the two main browsers.

Setting this value in the `onMouseOver` event handler requires that you signal the browser to not immediately update the status bar value when the handler exits. To do this, return a Boolean `true` value to inhibit any further browser activity. Likewise, you'll also need to do this in a `onMouseMove` handler. The example illustrates a simple "mouse'O'meter" to display the current mouse coordinates as the mouse moves. This only works in MSIE, however, and then only while the window has focus. This might be useful when working out where to position objects with dynamic HTML.

Frame-set contents that run script to set status bar values might operate better if you explicitly tell them to set the status property that belongs to the top level window. That is accessible via the top property, thus:

```
top.status = "ABCD";

window.top.status = "ABCD";

top.defaultStatus = "ABCD";

window.top.defaultStatus = "ABCD";
```

Warnings:

- Beware that some browsers do not correctly restore the status bar to its default condition after the rollover is deactivated by moving the mouse pointer somewhere else. This is the case for Netscape 2 and 3 on the Windows platform. Adding an `onMouseOut` handler may provide a satisfactory work around.

Example code:

```
<BODY onMouseMove="mouseOmeter();">
<SCRIPT>
// An example mouse coordinate display for MSIE


function mouseOmeter()
{
   window.status = window.event.x + "," + window.event.y;
   return true;
}
```

See also:	`Frame` object, `onMouseOut`, `onMouseOver`, Status line, `Window` object, `Window.defaultStatus`

Window.statusbar (Property)

A reference to an object that represents the status bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`Bar object`	
JavaScript syntax:	-	*myWindow*`.statusbar`
	-	`statusbar`

This is a read-only property containing a reference to a `Bar` object whose `visible` property contains a Boolean value that controls the visibility of the screen furniture represented by the object, in this case, the status bar.

Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to change the visibility of the status bar.

Example code:

```
// Request necessary privileges
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
// Hide status bar
window.statusbar.visible = false;
// There is another way that works without requesting privilege
// window.open('', '_top', 'status=0');
```

See also:	`Bar` object, `Frame` object, `UniversalBrowserWrite`, Window furniture, `Window` object, `Window.locationbar`, `Window.menubar`, `Window.personalbar`, `Window.scrollbars`, `Window.toolbar`

Property attributes:

`ReadOnly`.

Window.stop() (Method)

This duplicates the behavior of the Stop button on the Netscape Navigator button bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	`undefined`	
JavaScript syntax:	-	*myWindow*`.stop()`
	-	`stop()`

This is a means of immediately halting the script being executed. It's likely this will mainly be used in an error handler, although you may want to conditionally prevent a form from being submitted.

It is effectively the same as clicking on the Stop button in the toolbar.

Warnings:

- Since this is only supported on Netscape Navigator, it is not a portable feature and not recommended for deployment.

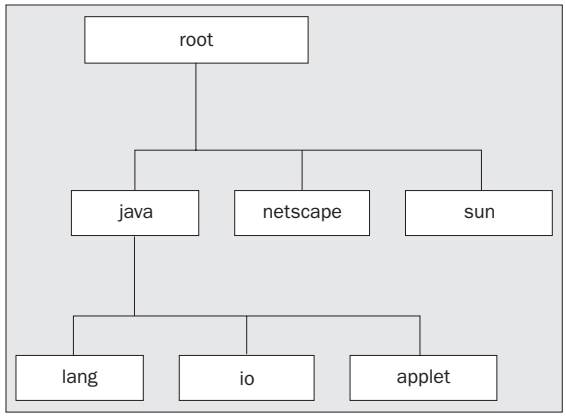
See also:	`Frame` object, `Window` object, `Window.find()`, `Window.home()`, `Window.print()`

Window.sun (Property)

A reference to the Java package object that is the root of the 'sun.*' Packages tree.

Availability:	JavaScript - 1.1 Netscape - 3.0	
Property/method value type:	JavaPackage sun	
JavaScript syntax:	N	*myWindow*.Packages.sun
	N	*myWindow*.sun
	N	Packages.sun
	N	sun

The object referred to by this property sits at the top of the sun package name hierarchy. It is through this property that you can access the java objects, properties and methods via LiveConnect.



See also:	JavaPackage object, Window.java, Window.netscape, Window.Packages

Property attributes:

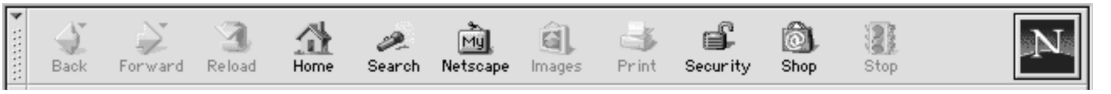
ReadOnly.

Window.toolbar (Property)

A reference to an object that represents the tool bar.

Availability:	JavaScript - 1.2 Netscape - 4.0	
Property/method value type:	Bar object	
JavaScript syntax:	-	*myWindow*.toolbar
	-	toolbar

This is a read-only property containing a reference to a `Bar` object whose `visible` property contains a Boolean value that controls the visibility of the screen furniture represented by the object, which, in this case, is the toolbar.



Warnings:

- Your script will need to be granted the `UniversalBrowserWrite` privilege to allow it to change the visibility of the toolbar.
- Be careful if you set both the menubar and toolbar to invisible, you will then have no reload capability to be able to refresh the screen. You will then only be able to quit.
- In the Netscape Navigator 4 browser for Macintosh, setting the toolbar invisible and then visible restores the tool bar but hides the menu items. On that platform, hiding the toolbar also hides all the other window furniture such, as locationbar and statusbar.

Example code:

```
// Request necessary
privilegesnetscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrit
e");// Hide tool barwindow.toolbar.visible = false;// There is another way that
works without requesting privilegewindow.open('', '_top', 'toolbar=0');
```

See also:	`Bar` object, `Frame` object, `UniversalBrowserWrite`, Window furniture, `Window` object, `Window.locationbar`, `Window.menubar`, `Window.personalbar`, `Window.scrollbars`, `Window.statusbar`

Property attributes:

`ReadOnly`.

Window.top (Property)

The topmost window in a framed hierarchy.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*myWindow*`.top`
	-	`top`

The `top` property should consistently refer to the window at the top of the hierarchy. The `window` property refers to the current window.

You can test attributes of the top window against attributes of the current window to see if your script is running in a correctly framed context. This may be useful if you have a frame dependent page that may have been linked to by a search engine. You can force the page into the correct frame-set if you can detect that it has been invoked outside of it.

```
if(top != window)
{
// The window is in a frameset
}
```

This property will contain a meaningful value regardless of whether the window is in a frame or not.

If the pages are known to come from the same server, you may want to check the `window.location.href` property against the `top.location.href property`. However, this may cause problems if the pages come from different servers and cause the script to throw a security related exception.

Warnings:

- Note that this value is not necessarily the same as the `parent` property.

See also:	`Frame` object, `Window` object, `Window.frame`, `Window.frames[]`, `Window.parent`, `Window.window`

Property attributes:

`ReadOnly`.

Window.window (Property)

Another name for the `self` property.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Opera - 3.0	
Property/method value type:	`Window object`	
JavaScript syntax:	-	*myWindow*`.window`
	-	`window`

This is another name for the `window.self` property in this context. However, `window` is useful because you can remove the ambiguity associated with accessing global object properties and methods. Don't forget that this can also refer to a frame as well as a window since they are both represented by the `window` object.

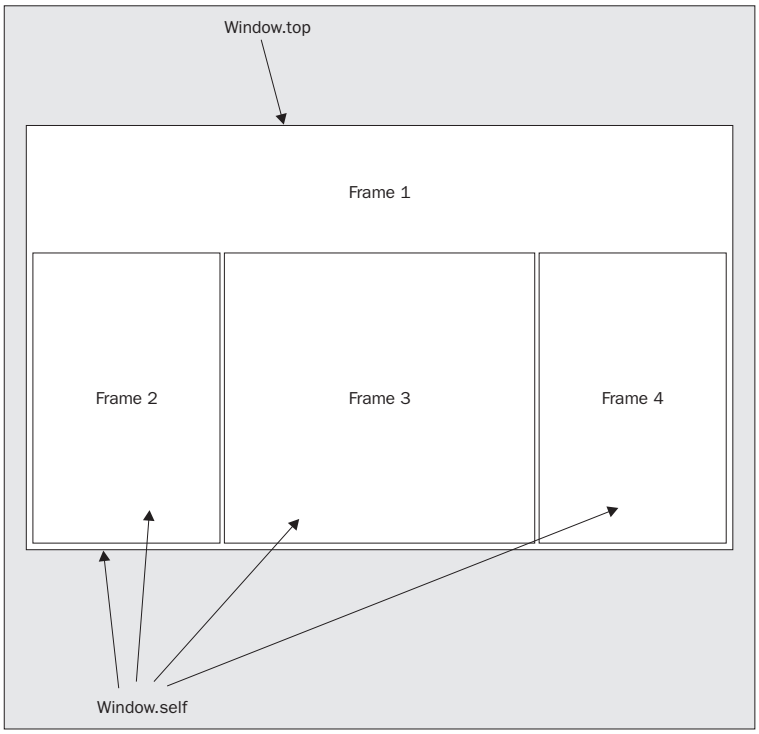
If the script is running in the topmost window of a frame-set or within a non-frame-set window, the top property will also be set to the same value.

This property is slightly odd, because the `window` object is also the global object for a web browser. The global object is always placed in the scope chain for a script's execution context. So, all properties that belong to the window are available without requiring the `window` object prefix.

So, these all refer to the same thing:

```
window

window.self

self

window.window

self.self

self.window.self

window.window.window.window.window.self.window
```

The main use of this property is to yield slightly better readability and clarity in the code you write. The same applies to the `self` property. It is better to explicitly call `window.open()` rather than just `open()`. Explicitly calling `window.open()` avoids an inadvertent call to `document.open()`. You might not be aware of an `open()` method belonging to another object that has been placed into the scope chain ahead of the `window` object.



Warnings:

- Be careful if you are building recursive scripts to walk the window hierarchy, as you could find yourself in an endless loop simply walking via the `window` property of the top level window.

See also:	`Frame` object, `Window` object, `Window.frame`, `Window.self`, `Window.top`

Property attributes:

`ReadOnly`.

Windows Script Host (Product)

A scripting environment available on the Windows platform.

On the Apple Macintosh, the AppleScript operating system extension has provided a way for applications to exploit AppleEvents and execute script-based control over one another.

On Windows, Microsoft introduced Windows Script Host (WSH), which provides a script-driven interface to the underlying COM model. This is a great improvement on the DOS batch commands that were available prior to this. Its still doesn't integrate applications in the way that AppleScript does, instead it integrates data objects with one another, which in some ways may be more powerful as it is a document-centric system. However, it can be difficult to exploit specific capabilities of applications unless they are COM related.

Like AppleEvents, WSH is language independent and you have a choice of languages that you can use for scripting. JScript is becoming more favored lately, although much has been made of VBScript, although it's not as powerful as Visual Basic, on which it is based.

For an in-depth discussion on WSH, consult the Wrox Professional JavaScript book, where a whole chapter is devoted to WSH.

with ... (Statement)

Adds an object to the front of the scope chain for use in the following block of script code.

Availability:	ECMAScript edition - 2 JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.02 Netscape - 2.0 Netscape Enterprise Server - 2.0 Opera - 3.0	
JavaScript syntax:	-	`with( anObject ) { someCode };`
Argument list:	*someCode*	Some code to execute with the enhanced scope chain
	anObject	A reference to an object to add to the scope chain

This statement is provided as a convenience mechanism to simplify your script code and save time and trouble.

When a statement executes, your line of script is running in one context or another. Each context is created and destroyed as functions are called and exit respectively. The contexts are added to an inheritance chain, which allows variables to be shared globally or locally.

The `with` keyword adds an object to the front of the scope chain for the current execution context. This saves you having to describe the full object reference since it is placed in the scope chain and always available and implicitly provided by JavaScript when resolving references to identifiers.

The code in the statement block is then executed while this augmented scope chain is in place. The object that is added to the scope chain is computed by the expression value in parentheses.

When the statement block is executed, the scope chain is restored to its original condition. This happens regardless of how the statement block is completed. Although the ECMA standard is ambiguous on this point, the implication is that a `break` might be appropriate in this context. A `continue` or `return` in the statement block would be inappropriate unless perhaps the `with()` construct is used within an iterator or function. Because of this ambiguity, you may find this behaves differently according to the implementation you are using.

The `with` statement can save you effort typing object names over and over again.

Warnings:

- Even though it is very convenient, it is considered somewhat bad form to use this construct. The code is hard to optimize inside the interpreter, which means it will likely run more slowly. Functions and variables instantiated inside a `with` block do not behave consistently, so it is recommended that you avoid using this construct.

Example code:

```
// Create a new object
var myObject = new Object;
// Add a property containing another object
myObject.itsObject = new Object;
// Add a property to that object
myObject.itsObject.someProperty = "String text";
// Now enhance the scope chain
with(myObject.itsObject)
{
   document.write(someProperty);
}
```

See also:	Identifier, Identifier resolution, Scope chain, Statement

Cross-references:

ECMA 262 edition 2 – section - 10.1.4

ECMA 262 edition 2 – section - 12.10

ECMA 262 edition 3 – section - 10.1.4

ECMA 262 edition 3 – section - 12.10

Wrox *Instant JavaScript* – page - 35

WML (Standard)

Wireless Mark-up Language.

This is the markup language used to describe cards (analogous to pages) in a wireless mobile device. This is the framework in which the WScript code will run. This is variously referred to as WScript and WMLScript and should not be confused with the WScript that Microsoft refer to as being a component of WSH.

See also:	Interpret, WAP, WScript

WScript (Standard)

Otherwise known as WMLScript or WAP Script – a variation of JavaScript for use in mobile devices.

This is the somewhat modified version of JavaScript, which is used in WAP mobile devices such as cellphones.

Warnings:

- Be aware that Microsoft have implemented a WScript object as part of the WSH environment. That object has absolutely nothing to do with WAP or mobile telecommunications. It is a container for an object model that is fundamental to WSH.

See also:	Host environment, Interpret, Platform, Script execution, WAP, WML

WScript object (Object/WSH)

An object that represents the object model of the WSH framework.

Availability:	JScript - 3.0	
JavaScript syntax:	WSH	`WScript`
Object properties:	`Application, Arguments, FullName, Name, Network, Path, ScriptFullName, ScriptName, StdErr, StdIn, StdOut, Version`	
Object methods:	`CreateObject(), DisconnectObject(), Echo(), GetObject(), Quit(), Sleep()`	

Property	JavaScript	JScript	N	IE	Opera	Notes
`Application`	-	3.0 +	-	-	-	-
`Arguments`	-	3.0 +	-	-	-	-
`FullName`	-	3.0 +	-	-	-	-
`Name`	-	3.0 +	-	-	-	-
`Network`	-	3.0 +	-	-	-	-

Table continued on following page

Property	JavaScript	JScript	N	IE	Opera	Notes
Path	-	3.0 +	-	-	-	-
ScriptFullName	-	3.0 +	-	-	-	-
ScriptName	-	3.0 +	-	-	-	-
StdErr	-	3.0 +	-	-	-	-
StdIn	-	3.0 +	-	-	-	-
StdOut	-	3.0 +	-	-	-	-
Version	-	3.0 +	-	-	-	-

Method	JavaScript	JScript	N	IE	Opera	Notes
CreateObject()	-	3.0 +	-	-	-	-
DisconnectObject()	-	3.0 +	-	-	-	-
Echo()	-	3.0 +	-	-	-	-
GetObject()	-	3.0 +	-	-	-	-
Quit()	-	3.0 +	-	-	-	-
Sleep()	-	3.0 +	-	-	-	-

WScript.Application (Property)

Access to the IDispatch interface for the object.

Availability:	JScript - 3.0	
Property/method value type:	IDispatch object	
JavaScript syntax:	WSH	*myApplication* = WScript.Application

This mechanism describes how to gain access to the objects, properties, and methods belonging to an external application.

See also:	WScript.GetObject()

WScript.Arguments (Property)

This returns a collection of argument items.

Availability:	JScript - 3.0	
Property/method value type:	WshArguments object	
JavaScript syntax:	WSH	myArguments = WScript.Arguments

The arguments are passed to the WSH script when it is executed. This collection provides a way to access the arguments, possibly by way of an enumerator.

Arguments from the command line or specified by the shortcut that executed the script are passed by means of this collection.

WScript.CreateObject() (Method)

Creates an instance of an automation object.

Availability:	JScript - 3.0	
Property/method value type:	`WScript object`	
JavaScript syntax:	WSH	*`myObject`* `= WScript.CreateObject(`*`aProgID`*`);`
	WSH	*`myObject`* `= WScript.CreateObject(`*`aProgID`*`,` *`aPrefix`*`);`
Argument list:	*`aProgID`*	An application programme ID
	`aPrefix`	A hook into the event model

Calling this method yields an object which can be used to communicate with another application residing in the same computer.

For example, this code creates an object that references the Word application:

```
myWord = WScript.CreateObject("Word.Application");
```

From here we can reference the application via the object. This makes the Word application we just instantiated visible to the user:

```
myWord.Visible = true;
```

WScript.DisconnectObject() (Method)

Discards an object.

Availability:	JScript - 3.0	
JavaScript syntax:	WSH	`WScript.DisconnectObject(`*`anObject`*`)`
Argument list:	*`anObject`*	An object previously created by WSH

This method is used to get rid of objects previously created with the `CreateObject()` and `GetObject()` methods.

WScript.Echo() (Method)

Echoes some output to the caller via standard output.

Availability:	JScript - 3.0	
Property/method value type:	`WScript object`	
JavaScript syntax:	WSH	`WScript.Echo(`*`anArg`*`)`
	WSH	`WScript.Echo(`*`anArg`*`, ...)`
Argument list:	*`anArg`*	A string or numeric value

This works just like the `echo` command in a shell scripting environment.

It takes a variable number of arguments which can be strings or numbers.

The behavior depends on which interpreter is being used. In the cscript interpreter, the results are concatenated together into the output stream. In the wscript interpreter, each `Echo()` call results in an `alert()` dialog box which must be manually dismissed.

WScript.FullName (Property)

The full path and name for the file being executed.

Availability:	JScript - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	WSH	*`myName`* `= WScript.FullName`

This property contains the full name and path for the file being executed.

See also:	`WScript.ScriptFullName`

WScript.GetObject() (Method)

Access an already existing object rather than creating a new one.

Availability:	JScript - 3.0	
Property/method value type:	`WScript object`	
JavaScript syntax:	WSH	`WScript.GetObject(`*`aPath`*`)`
	WSH	`WScript.GetObject(`*`aPath`*`, `*`aProgID`*`)`
	WSH	`WScript.GetObject(`*`aPath`*`, `*`aProgID`*`, aPrefix)`

Argument list:	*aPath*	The path to an already existing document
	aProgID	An application programme ID
	aPrefix	A hook into the event model

You can use this method to access an object that you know already exists without needing to create a fresh instance. An example of the difference is that `CreateObject()` is used to manufacture new documents via the application while `GetObject()` is used to access existing documents.

Given the object that encapsulates the document, since it is a WScript object, you can access its `Application` property to control the owning application.

See also:	`WScript.Application`

WScript.Name (Property)

A human readable name for the script.

Availability:	JScript - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	WSH	*myName* `= WScript.Name`

Computers are happy to use arcane names for scripts but WSH provides a way to give your script a human-friendly name. This property returns such a value.

WScript.Network (Property)

A reference to a network management object.

Availability:	JScript - 3.0	
Property/method value type:	`WshNetwork object`	
JavaScript syntax:	WSH	*myNet* `= WScript.Network`

With the object returned by this property, you can manage the network within the computer, mapping in new drives or unmapping existing ones, for instance.

WScript.Path (Property)

The path to the activating WSH executive.

Availability:	JScript - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	WSH	*myPath* `= WScript.Path`

This property yields a string that describes which, of several, WSH run-time environments is used.

WScript.Quit() (Method)

Terminates the script and returns an error code.

Availability:	JScript - 3.0	
JavaScript syntax:	WSH	`WScript.Quit(`*`anErrCode`*`)`
Argument list:	*`anErrCode`*	An error code to be sent back to the caller

The current instance of cscript or wscript is killed and an error returned to the calling shell.

WScript.ScriptFullName (Property)

The full path and script name.

Availability:	JScript - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	WSH	*`myName`* `= WScript.ScriptFullName`

With this property, you can dismantle the path to the file and generate relative paths that locate temporary files adjacent to the script that is being executed. If the script is moved, the relative locations move with it.

See also:	WScript.FullName

WScript.ScriptName (Property)

A string containing the name of the script.

Availability:	JScript - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	WSH	*`myName`* `= WScript.ScriptName`

With this property, you can create scripts that will behave differently according to the name under which they are executed.

WScript.Sleep() (Method)

Suspend the script execution for a while.

Availability:	JScript - 3.0	
JavaScript syntax:	WSH	`WScript.Sleep(aDuration)`
Argument list:	*aDuration*	A value specified in milliseconds

With this method, you can suspend execution for a while without creating a 'busy waiting' loop.

WScript.StdErr (Property)

A write-only stream used for output.

Availability:	JScript - 3.0	
Property/method value type:	`TextStream object`	
JavaScript syntax:	WSH	*myStream* `= WScript.StdErr`

This is only available to WSH scripts being executed from a command line interface. It provides a way for the scripts to communicate error messages back to the caller.

WScript.StdIn (Property)

A read-only stream used for input.

Availability:	JScript - 3.0	
Property/method value type:	`TextStream object`	
JavaScript syntax:	WSH	*myStream* `= WScript.StdIn`

This is only available to WSH scripts being executed from a command line interface. It provides a way for the scripts to receive messages input from the caller.

WScript.StdOut (Property)

A write-only stream used for output.

Availability:	JScript - 3.0	
Property/method value type:	`TextStream object`	
JavaScript syntax:	WSH	*myStream* = `WScript.StdOut`

This is only available to WSH scripts being executed from a command line interface. It provides a way for the scripts to communicate messages back to the caller.

WScript.Version (Property)

A string containing the WSH version number.

Availability:	JScript - 3.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	WSH	`myVersion = WScript.Version`

With this, you can write version dependent code.

WSH (Object model)

The object model used in Windows Script Host.

See also:	WScript.GetObject(), `WScript` object

WSH (Product)

Windows Script Host.

See also:	`.htc`, Scriptlet, Web browser, Windows Script Host

wysiwyg: (Request method)

Special URL method to handle page content when resized in Netscape Navigator.

If a page is generated using JavaScript, then if the page is subsequently resized, this special method is used to encapsulate the previous page location and invoke special handling to ensure that the page is printed properly.

XML (Standard)

Extensible Mark-up Language.

This is gradually becoming commonplace as a way to exchange data between systems. It is also supported by the MSIE browser and you can load in XML documents directly.

It is the future direction that HTML will evolve towards, beginning with XHTML.

This topic is the entry point to complete new subject area. Its too vast to attempt to cover it meaningfully in just a few pages and yet it's probably going to become one of the most important parts of the web programming landscape.

Refer to the Wrox XML reference manual for details of how to use it in earnest. Here we will just scratch the surface to begin to see what it looks like.

This creates a new XML document via ActiveX:

```
myXMLDoc = new ActiveXObject("Microsoft.XMLDOM");
```

Having created it, now we can load the contents of a URL into the object:

```
myXMLDoc.load("http://xmlserver.domain.com/reports.xml");
```

This is useful because one big problem with JavaScript in a web browser is that it's very hard to download a data file from a web server to a script without having to work around lots of security issues. This might solve that problem a lot more elegantly.

Now we can begin to look at the contents of the file and extract information from it.

```
myXMLDoc.loadXML("JoeSmith");
```

Now that we have acquired the document we can use the DOM navigation techniques that already work for HTML documents to walk through the XML structures:

```
var myXMLNodeList = myXMLDoc.getElementsByTagName(strNodeName);
```

XML name (Definition)

A strict convention for naming items within an XML compliant document with a well formed token.

An XML name may start with a letter, underscore or colon. However the colon is reserved for use with namespaces and is not fully standardized as of XML version 1.

Subsequent characters in the name may be any one of:

- Letter
- Digit
- Dot (period/full stop)
- Dash (minus sign)
- Underscore
- Colon
- CombiningChar
- Extender

Your name values should not begin with the string "XML" or any sequence of characters that might degenerate to it after case conversion or international character substitution.

Combining characters and Extenders are enumerated in Appendix B of the XML version 1.0 standard.

See also:	`Event.type`

XML object (Object/JScript)

An object that represents a block of XML page content within an HTML document.

Availability:	JScript - 5.0 Internet Explorer version - 5.0		
Inherits from:	`Element` object		
JavaScript syntax:	IE	`myXML = myDocument.all.anElementID`	
	IE	`myXML = myDocument.all.tags("XML")[anIndex]`	
	IE	`myXML = myDocument.all[aName]`	
	-	`myXML = myDocument.getElementById(anElementID)`	
	-	`myXML = myDocument.getElementsByName (aName)[anIndex]`	
	-	`myXML = myDocument.getElementsByTagName ("XML")[anIndex]`	
Argument list:	*`anIndex`*	A reference to an element in a collection	
	`aName`	An associative array reference	
	`anElementID`	The ID value of an Element object	

Object properties:	canHaveHTML, defer, event, htmlFor, src, text, type
Event handlers:	onDataAvailable, onDatasetChanged, onDatasetComplete, onReadyStateChange, onRowEnter, onRowExit, onRowsDelete, onRowsInserted

The MSIE browser can now cope with pages delivered as arbitrary blocks of XML. If it encounters an <XML> tag, then it will instantiate one of these objects to provide JavaScript binding to it. This can also be used to build a small island of XML based content in the middle of an HTML page.

The XML data can sit in an HTML page like this:

```
<XML ID="myBlock">

<METADATA>

<OWNER>Wrox</OWNER>

<DATATYPE>Example</DATATYPE>

<ABSTRACT>This is an example block of text</ABSTRACT>

</METADATA>

</XML>
```

Accessing the text property of the XML object will return all the inner text inside it. To access the components you will need to access the XMLDocument property to expose a DOM document interface. This can be explored using DOM compatible methods and properties.

See also:	Document.readyState, onRowEnter, onRowExit

Property	JavaScript	JScript	Nav	IE	Opera	Notes
canHaveHTML	-	5.0 +	-	5.0 +	-	-
defer	-	5.0 +	-	5.0 +	-	-
event	-	5.0 +	-	5.0 +	-	-
htmlFor	-	5.0 +	-	5.0 +	-	-
src	-	5.0 +	-	5.0 +	-	-
text	-	5.0 +	-	5.0 +	-	-
type	-	5.0 +	-	5.0 +	-	-
onDataAvailable	-	3.0 +	-	4.0 +	-	-
onDatasetChanged	-	3.0 +	-	4.0 +	-	-

Event name	JavaScript	JScript	Nav	IE	Opera	Notes
onDatasetComplete	-	3.0 +	-	4.0 +	-	-
onReadyStateChange	-	3.0 +	-	4.0 +	-	-
onRowEnter	-	3.0 +	-	4.0 +	-	-
onRowExit	-	3.0 +	-	4.0 +	-	-
onRowsDelete	-	3.0 +	-	4.0 +	-	-
onRowsInserted	-	3.0 +	-	4.0 +	-	-

Inheritance chain:

`Element` object, `Node` object

Web-references:

http://msdn.microsoft.com/xml/xmlguide/dom-guide-document.asp

XML.defer (Property)

A property containing a deferral status for the XML block.

Availability:	JScript - 5.0 Internet Explorer version - 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myXML.defer`

This can defer the processing of an XML object until later in the page loading and display process.

See also:	SCRIPT.defer

XML.event (Property)

An event object associated with the XML block.

Availability:	JScript - 5.0 Internet Explorer version - 5.0	
Property/method value type:	`Event object`	
JavaScript syntax:	IE	`myXML.event`

Events can be associated with an XML object in much the same way that they would be used with others. The model is slightly different because it is more generalised.

The Microsoft documentation lists these event handlers as being appropriate for an XML object:

- `onDataAvailable`
- `onDatasetChanged`
- `onDatasetComplete`
- `onReadyStateChange`
- `onRowEnter`
- `onRowExit`
- `onRowsDelete`
- `onRowsInserted`

See also:	`onDataAvailable, onDataSetChanged, onDataSetComplete, onReadyStateChange, onRowEnter, onRowExit, onRowsDelete, onRowsInserted`

XML.src (Property)

The URL where the contents of the XML block are to be loaded from.

Availability:	JScript - 5.0 Internet Explorer version - 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myXML.src`

You may be able to redefine this property value and reload the XML block from a different source.

XML.text (Property)

The textual content of the XML block.

Availability:	JScript - 5.0 Internet Explorer version - 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myXML.text`

Any textual content will be yielded up by this property, because XML support is still evolving this property and the XML object may change.

For now, you will get all the text contained within the node and its children.

This means that for the example:

```
<XML ID="myBlock">
<METADATA>
<OWNER>Wrox</OWNER>
<DATATYPE>Example</DATATYPE>
<ABSTRACT>This is an example block of text.</ABSTRACT>
</METADATA>
</XML>
```

The text for the top level object will be:

Wrox Example This is an example block of text.

The text for the object presented by the XMLDocument property will be the same as the text for the contained <METADATA> node.

The text for <OWNER>, <DATATYPE> and <ABSTRACT> nodes will be their individual content.

This means it is important to walk down the tree to the node or group of nodes you want before trying to extract the text.

XML.type (Property)

The MIME type of the XML data file.

Availability:	JScript - 5.0 Internet Explorer version - 5.0	
Property/method value type:	`String primitive`	
JavaScript syntax:	IE	`myXML.type`

The MIME type of the document associated with the XML object is accessible through the value of this property.

Refer to the MIME type topic for details of the available MIME types you will likely see in this property.

See also:	MIME types

XML.XMLDocument (Property)

A reference to the top of a DOM hierachy that describes the content of the XML data island.

Availability:	JScript - 5.0 Internet Explorer version - 5.0	
JavaScript syntax:	IE	*myXML*.XMLDocument

Given the example of a block of XML in an HTML document:

```
<XML ID="myBlock">

<METADATA>

<OWNER>Wrox</OWNER>

<DATATYPE>Example</DATATYPE>

<ABSTRACT>This is an example block of text.</ABSTRACT>

</METADATA>

</XML>
```

individual nodes in that so called data island can be accessed through this XMLDocument property. The object returned by this property responds to the `selectSingleNode()` method. The argument to this is the slash separated path to the node within the document you are looking for. The slash separated values are the XML tagnames used to construct the document.

In this example, they all begin with the string `"METADATA"` and since the document only contains one layer inside that, all nodes can be reached with the following strings:

- `METADATA/OWNER`
- `METADATA/DATATYPE`
- `METADATA/ABSTRACT`

Given that our XML block has an ID value of "`myBlock`" this line of script code should yield a reference to an object that encapsulates the `<ABSTRACT>` node:

```
myBlock.XMLDocument.selectSingleNode("METADATA/ABSTRACT")
```

Having accessed the DOM node you want, its content can be examined by looking at its text property.

The example code illustrates this concept as it might be assembled together in a simple form.

Warnings:

- This property returns an undefined value in the Macintosh version of MSIE 5.0 instead of a reference to a DOM document.

Example code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>

<!-- Create an XML island -->
<XML ID="myBlock">
   <METADATA>
      <OWNER TYPE="PUBLISHER">Wrox</OWNER>
      <DATATYPE>Example</DATATYPE>
      <ABSTRACT>This is an example block of text.</ABSTRACT>
   </METADATA>
```

```
</XML>

<SCRIPT>
// Get the DOM document
myXMLDocument = myBlock.XMLDocument;

// Find the node
myNode = myXMLDocument.selectSingleNode("METADATA/ABSTRACT");

// Display the text in the node
alert(myNode.text);

// Now access node attributes and content
myOwner = myXMLDocument.getElementsByTagName('OWNER')[0];
alert(myOwner.getAttribute('TYPE') + ': ' + myOwner.text);
</SCRIPT>
</BODY>
</HTML>
```

XMP object (Object/HTML)

A deprecated object that has been replaced by the PRE object.

Availability:	JavaScript - 1.0 JScript - 1.0 Internet Explorer - 3.0 Netscape - 2.0 Deprecated	
JavaScript syntax:	IE	`myXMP = myDocument.all.anElementID`
	IE	`myXMP = myDocument.all.tags("XMP")[anIndex]`
	IE	`myXMP = myDocument.all[aName]`
	-	`myXMP = myDocument.getElementById(anElementID)`
	-	`myXMP = myDocument.getElementsByName(aName)[anIndex]`
	-	`myXMP = myDocument.getElementsByTagName ("XMP")[anIndex]`
HTML syntax:	`<XMP>`	
Argument list:	*anIndex*	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The ID value of an Element object
See also:	PRE object	

XRay() (Filter/visual)

A visual filter that displays only the element edges.

Availability:	JScript - 3.0 Internet Explorer version - 4.0
See also:	filter - `XRay()`

Year from time (Time calculation)

A date and time algorithm defined as part of ECMAScript.

Availability:	ECMAScript edition –2
Property/method value type:	`Number primitive`

Given a time value, it is helpful to be able to compute a year number from it. The ECMA compliant implementations use the extrapolated Gregorian system to map days to years and compute a time relative to the reference date of 01-January-1970 UTC. From there the year number can be established.

All non-leap years have 365 days with the usual number of days in each month. Leap years have an extra day in February. The calculation shown below uses known leap years and non-leap years to adjust the day numbers and yield the day number of the first day of the given year and then use that to work out the time in milliseconds when the year started:

```
DayFromYear(y) =

365 * (y - 1970) +

floor((y - 1969) / 4) -

floor((y - 1901) / 100) +

floor((y - 1601) / 400)

msPerDay = 86400000

TimeFromYear(y) = msPerDay * DayFromYear(y)

YearFromTime(t) = The largest integer y to make TimeFromYear(y) less than or equal
to t.
```

Example code:

```
<HTML>
<BODY>
<SCRIPT>
// Work out year number from time
var msPerDay = 86400000;
var myMilliseconds = Number(new Date());
document.write(yearFromTime(myMilliseconds));
// Return year number based on time value
function yearFromTime(aMilliseconds)
{
   var myStartYear = 1970;

   while(timeFromYear(myStartYear) < myMilliseconds)
   {
      myStartYear++
   }

   return myStartYear-1;
}

// Work out milliseconds at start of year
function timeFromYear(aYear)
{
   var myTime = msPerDay * dayFromYear(aYear);
   return myTime;
}

// Work out day number from milliseconds
function dayNumber(aMillisecondTime)
{
   var myDay = Math.floor(aMillisecondTime/msPerDay);

   return myDay;
}

// Day from year function
function dayFromYear(aYear)
{
   var myDay = 365 * (aYear - 1970)             +
   Math.floor((aYear - 1969) / 4)   -
   Math.floor((aYear - 1901) / 100) +
   Math.floor((aYear - 1601) / 400);
   return myDay;
}
</SCRIPT>
</BODY>
</HTML>
```

See also: Broken down time, Date from time, Date number, Day from year, Day within year, In leap year, Month from time, Time from year, Year number

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.3

ECMA 262 edition 3 – section – 15.9.1.3

Year number (Time calculation)

A date and time algorithm defined as part of ECMAScript.

Availability:	ECMAScript edition – 2
Property/method value type:	`Number primitive`

In calculating year numbers, ECMA compliant implementations should use an extrapolated Gregorian system. This allows for leap years and non leap years at the turn of the century, 3 out of every four.

There are various calculations related to year numbers. Refer to the various example calculations:

- Days in year
- Day from year
- Time from year
- Year from time
- In leap year

See also:	Broken down time, Days in year, In leap year, Time from year, Year from time

Cross-references:

ECMA 262 edition 2 – section – 15.9.1.3

ECMA 262 edition 3 – section – 15.9.1.3

Zero value (Definition)

ECMAScript interpreters must be able to distinguish between +0 and -0.

Availability:	ECMAScript edition - 2
Property/method value type:	`Number primitive`

In the real world, Zero is Zero is Zero. Inside a JavaScript interpreter, certain calculations (most notably `Math.pow()` functions) require the sign of the zero value to be taken into account.

Internally, an ECMA compliant interpreter must be able to distinguish between +0 and -0 although from a mathematical standpoint the values are identical. Where calculations involve the likelihood of +infinity and -infinity, the sign can affect the outcome and so can the sign of the zero value. For example the reciprocal of a very small number.

We have quietly ignored the sign of a zero value where it would make no difference. There are some discussions where it becomes important and so the sign is retained here for illustrative purposes.

See also:	Infinity, Math.pow()

Cross-references:

ECMA 262 edition 2 - section - 8.5

ECMA 262 edition 3 - section - 8.5

Zigzag() (Filter/transition)

Reveals the new image with a zigzag effect.

Availability:	JScript - 5.5 Internet Explorer version - 5.5

Refer to:

filter - `Zigzag()`

Symbols

! (Logical NOT) (Operator/logical)

Logical NOT operator.

Availability:	ECMAScript edition –2 JavaScript –1.0 JScript –1.0 Internet Explorer –3.02 Netscape –2.0 Netscape Enterprise Server –2.0 Opera –3.0
Property/method value type:	`Boolean primitive`

See also:	Bitwise `NOT` –complement (~), Logical operator, `NOT` Equal to (`!=`), Logical NOT –complement (`!`), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 –section –11.4.9

ECMA 262 edition 3 –section –11.4.9

!= (NOT equal) (Operator/equality)

Inequality operator.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`
See also:	Equal to (==), Equality operator, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), Logical NOT – complement (!), NOT Equal to (!=), NOT Identically equal to (!==), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.9.2

ECMA 262 edition 3 – section – 11.9.2

!== (NOT identical) (Operator/identity)

Compares two values for non-equality and identical type.

Availability:	ECMAScript edition – 3 JavaScript – 1.3 JScript – 1.0 Internet Explorer – 3.02 Netscape – 4.06
Property/method value type:	`Boolean primitive`
See also:	Equal to (==), Equality expression, Equality operator, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Identity operator, Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==)

Cross-references:

ECMA 262 edition 3 – section – 11.9.5

" (Double quote) (Delimiter)

Double quote string literal delimiter.

Availability:	ECMAScript edition – 2 Opera – 3.0
Property/method value type:	`String primitive`

Refer to:

String literal

Cross-references:

ECMA 262 edition 2 – section – 7.7.3

ECMA 262 edition 3 – section – 7.8.4

$ (Dollar) (Symbol)

A special character allowed to be used in an identifier name.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Opera – 3.0

Refer to:

Identifier

Cross-references:

ECMA 262 edition 2 – section – 7.5

ECMA 262 edition 3 – section – 7.6

$n (Numbered argument) (Property/static)

A property of the global `RegExp` object.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0	
JavaScript syntax:	-	`RegExp.$n`
Argument list:	*n*	An index number

Property attributes:

`ReadOnly`.

Refer to:

`RegExp.$n`

% (Modulo/remainder) (Operator/multiplicative)

Divides one operand by another and yields the remainder.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Divide (`/`), Multiplicative operator, Remainder (`%`), Remainder then assign (`%=`)

Cross-references:

ECMA 262 edition 2 – section – 11.5.3

ECMA 262 edition 3 – section – 11.5.3

%= (Modulo assign) (Operator/assignment)

Divides one operand by another and stores the result in the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Assignment operator, LValue, Multiplicative operator, Remainder (`%`), Remainder then assign (`%=`)

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

& (Bitwise AND) (Operator/bitwise)

Bitwise AND of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Binary bitwise operator, Bitwise AND (`&`), Bitwise AND then assign (`&=`), Logical AND (`&&`)

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.10

&& (Logical AND) (Operator/logical)

Logical AND of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`

See also:	Binary logical operator, Bitwise AND (`&`), Bitwise AND then assign (`&=`), Logical AND (`&&`)

Cross-references:

ECMA 262 edition 2 – section – 11.11

ECMA 262 edition 3 – section – 11.11

&= (Bitwise AND assign) (Operator/assignment)

Bitwise AND of two operands, and assigns the result to the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Assignment operator, Bitwise AND then assign (`&=`), Bitwise operator, Logical AND (`&&`), LValue

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

' (Single quote) (Delimiter)

Single quote string literal delimiter.

Availability:	ECMAScript edition – 2 Opera – 3.0
Property/method value type:	`String primitive`

See also:	ASCII, Escape sequence (`\`), Line terminator, Literal, Punctuator, String, String literal, Unicode, `var`

Cross-references:

ECMA 262 edition 2 – section – 7.7.3

ECMA 262 edition 3 – section – 7.8.4

( ) (Argument delimiter) (Delimiter)

Delimits the arguments of a function call.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	*`myFunction(someArguments)`*
Argument list:	*`someArguments`*	An optional collection of arguments to the function

See also:	Function, `function( ... ) ...`, Function call, Grouping operator `( )`, Operator, Operator Precedence, Parentheses `( )`, Postfix operator, Primary expression

Cross-references:

ECMA 262 edition 2 – section – 11.2

ECMA 262 edition 3 – section – 11.1.6

( ) (Grouping operator) (Delimiter)

Control of expression operator execution precedence and `Function` argument delimiters.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

See also:	Function, Function call, `function( ... ) ...`, Grouping operator `( )`, Operator, Operator Precedence, Parentheses `( )`, Postfix operator, Primary expression

Cross-references:

ECMA 262 edition 2 – section – 11.1.4

ECMA 262 edition 3 – section – 11.1.6

* (Multiply) (Operator/multiplicative)

Multiply one operand by another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

Property/method value type:	`Number primitive`

See also:	Multiplicative operator, Multiply (`*`)

Cross-references:

ECMA 262 edition 2 – section – 11.5.1

ECMA 262 edition 3 – section – 11.5.1

*/ (Close comment block) (Delimiter)

Closing token for a multi-line comment.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

See also:	Comment, Comment (`//` and `/* ... */`)

Cross-references:

ECMA 262 edition 2 – section – 7.3

ECMA 262 edition 3 – section – 7.4

*= (Multiply assign) (Operator/assignment)

Multiplies two operands and assigns the result to the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Assignment operator, LValue, Multiplicative operator, Multiply then assign (*=)

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

+ (Add) (Operator/additive)

Adds two operands together.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Add (+), Additive operator, Positive value (+), String concatenate (+), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.6.1

ECMA 262 edition 3 – section – 11.6.1

+ (Concatenate) (Operator/string)

Joins string operators by concatenation.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`String primitive`

See also:	Add (+), Additive operator, Positive value (+), `String.concat()`, String concatenate (+), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.6.1

ECMA 262 edition 3 – section – 11.6.1

+ (Unary plus) (Operator/unary)

Indicates positive value or numeric cast a non-numeric value.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Add (+), Additive operator, Positive value (+), String concatenate (+), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.4.6

ECMA 262 edition 3 – section – 11.4.6

++ (Post increment) (Operator/postfix)

Incrementing operator.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Decrement value (--), Increment value (++), Postfix increment (++), Prefix increment (++), Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.3.1

ECMA 262 edition 3 – section – 11.3.1

++ (Pre increment) (Operator/prefix)

Increments an operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server– 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Additive operator, Decrement value (--), Increment value (++), Postfix increment (++), Prefix increment (++), Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.4.4

ECMA 262 edition 3 – section – 11.4.4

+= (Add assign) (Operator/assignment)

Adds the second operand to the first, modifying the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Additive operator, Add then assign (+=), Assignment operator, Concatenate then assign (+=), Increment value (++), LValue

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

, (Comma) (Delimiter)

Comma separated argument list. The comma operator provides a way to evaluate several assignment expressions at once.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
See also:	Comma operator (,), `Document.write()`, `Document.writeln()`, `var`

Cross-references:

ECMA 262 edition 2 – section – 11.14

ECMA 262 edition 3 – section – 11.14

- (Minus) (Operator/additive)

Subtracts one operand from another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Additive operator, Negation operator (-), Subtract (-), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.6.2

ECMA 262 edition 3 – section – 11.6.2

- (Unary minus) (Operator/unary)

Negates the value of an operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server– 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Additive operator, Negation operator (-), Subtract (–), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.4.7

ECMA 262 edition 3 – section – 11.4.7

– (Post decrement) (Operator/postfix)

Decrementing operator.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Decrement value (--), Increment value (++), Postfix decrement (--), Prefix decrement (--), Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.3.2

ECMA 262 edition 3 – section – 11.3.2

– (Pre decrement) (Operator/prefix)

Decrements an operand's value.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Additive operator, Decrement value (--), Increment value (++), Postfix decrement (--), Prefix decrement (--), Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.4.5

ECMA 262 edition 3 – section – 11.4.5

-= (Minus assign) (Operator/assignment)

Subtracts the right value from the left modifying the left.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
See also:	Additive operator, Assignment operator, LValue, Subtract (–), Subtract then assign (–=)

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

. (Decimal point) (Delimiter)

A delimiter character.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
See also:	Floating point, Object property delimiter (.), Property accessor, Decimal point (.)

Cross-references:

ECMA 262 edition 2 – section – 7.7.3

ECMA 262 edition 3 – section – 7.8.3

. (Period) (Delimiter)

A token to delimit object properties from their object or a decimal point.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
See also:	Member, Object property delimiter (`.`), Property accessor

Cross-references:

ECMA 262 edition 2 – section – 8.6

ECMA 262 edition 2 – section – 11.2

ECMA 262 edition 3 – section – 11.2.1

/ (Divide) (Operator/multiplicative)

Divides one operand by another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Arithmetic operator, Divide (`/`), Multiplicative operator, Remainder (`%`)

Cross-references:

ECMA 262 edition 2 – section – 11.5.2

ECMA 262 edition 3 – section – 11.5.2

/ (Slash) (Delimiter)

A delimiter character for regular expressions.

Availability:	JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0

Refer to:

`RegExp` literal

/* ... */ (Comment block) (Delimiter)

A multi-line comment delimiting token.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
JavaScript syntax:	-	`/*someCommentText*/`
Argument list:	`someCommentText`	Any arbitrary comment text required

See also:	Comment, Comment (`//` and `/* ... */`), Multi-line comment

Cross-references:

ECMA 262 edition 2 – section – 7.3

ECMA 262 edition 3 – section – 7.4

/*@ ... @*/ (Pre processing block) (Delimiter)

A special form of the comment delimiters for enclosing pre-processor directives.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	-	`/*@someDirectives@*/`
Argument list:	*`someDirectives`*	One or more pre-processor directives

Refer to:

Pre-processing – `/*@ ... @*/`

// (Comment line) (Delimiter)

A single line comment delimiting token.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server– 2.0 Opera – 3.0
See also:	Comment, Comment (`//` and `/* ... */`), Single line comment

Cross-references:

ECMA 262 edition 2 – section – 7.3

ECMA 262 edition 3 – section – 7.4

/= (Divide assign) (Operator/assignment)

Divides one operand by another, leaving the result in the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Arithmetic operator, Assignment operator, Divide then assign (`/=`), LValue, Multiplicative operator, Remainder (`%`)

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

: (Colon) (Delimiter)

A delimiter used with labels and conditional operators.

Availability:	ECMAScript edition – 2 JavaScript – 1.2 JScript – 3.0 Internet Explorer – 4.0 Netscape – 4.0 Netscape Enterprise Server – 3.0
See also:	Colon (`:`), `switch( ... ) ... case: ... default: ...`

Cross-references:

ECMA 262 edition 2 – section – 7.4.3

ECMA 262 edition 3 – section – 7.5.2

; (Semicolon) (Delimiter)

Semicolon characters are used to separate one executable statement from another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
See also:	Empty statement (;), Expression statement, Punctuator, Semicolon (;), Statement

Cross-references:

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 2 – section – 12.3

ECMA 262 edition 2 – section – 12.4

ECMA 262 edition 3 – section – 12.2

ECMA 262 edition 3 – section – 12.3

ECMA 262 edition 3 – section – 12.4

< (Less than) (Operator/relational)

Compares two operands to determine which is nearer to `-Infinity`.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`

See also:	Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==), Relational operator

Cross-references:

ECMA 262 edition 2 – section – 11.8.1

ECMA 262 edition 3 – section – 11.8.1

<!– … –> (Comment block) (Object/HTML)

HTML comments can be used to hide scripts.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
Inherits from:	`CharacterData` object	
JavaScript syntax:	IE	*myDocument*`.all.tags("!")[`*anIndex*`]`
HTML syntax:	`<!-- `*someCommentText*` -->`	
Argument list:	*anIndex*	A reference to an element in a collection

See also:	`COMMENT` object, `Element` object, Hiding scripts from old browsers

Inheritance chain:

`CharacterData` object, `Node` object

<% … %> (Server side code block) (ASP tag)

This is a special tag to delimit server-side code to be executed in the ASP back end of an IIS server.

Refer to:

ASP

<< (Bitwise shift left) (Operator/bitwise)

Bitwise leftwards shift one operand according to another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift left (<<), Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.7.1

ECMA 262 edition 3 – section – 11.7.1

<<= (Bitewise shift left assign) (Operator/assignment)

Destructively bitwise leftwards shift the first of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Assignment operator, Bitwise operator, Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), LValue

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

<= (Less than or equal to) (Operator/relational)

Compares two operands to determine which is nearer to `-Infinity` or whether they are equal.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0	
Property/method value type:	`Boolean primitive`	
JavaScript syntax:	-	*`anOperand <= anOperand`*
Argument list:	*`anOperand`*	An operand that can be compared for magnitude

See also:	Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==), Relational operator

Cross-references:

ECMA 262 edition 2 – section – 11.8.3

ECMA 262 edition 3 – section – 11.8.3

= (Assign) (Operator/assignment)

Assigns the right value to the left operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server version – 2.0 Opera – 3.0
Property/method value type:	`Depends on right value`

Warnings:

- The operand to the left of the operator must be an `LValue`. That is, it should be able to take an assignment and store the value.
- Be careful not to confuse the single equals with the double equals. Placing a double equals in place of an assignment will not assign the value and may be considered to be a syntax error, since it may perform a comparison in entirely the wrong context. The interpreter may be forgiving enough that a run-time error isn't generated though, but the side effects could be subtle and make it hard to diagnose the cause.

See also:	Assign value (=), Assignment operator, Equal to (==), Equality operator, Lvalue, var, Variable statement

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 2 – section – 12.2

ECMA 262 edition 3 – section – 11.13

ECMA 262 edition 3 – section – 12.2

== (Equal to) (Operator/equality)

Compares two operands for equality.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`

See also:	= (Assign), Equality operator, Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==)

Cross-references:

ECMA 262 edition 2 – section – 11.9.1

ECMA 262 edition 2 – section – 11.9.3

ECMA 262 edition 3 – section – 11.9.1

ECMA 262 edition 3 – section – 11.9.3

=== (Identical to) (Operator/identity)

Compares two operands for equality and identical type.

Availability:	ECMAScript edition – 3 JavaScript – 1.3 JScript – 1.0 Internet Explorer – 3.02 Netscape – 4.06
Property/method value type:	`Boolean primitive`

See also:	Equal to (==), Equality expression, Equality operator, Greater than (>), Greater than or equal to (>=), Identically equal to (===), Identity operator, Less than (<), Less than or equal to (<=), NOT Equal to (!=), NOT Identically equal to (!==)

Cross-references:

ECMA 262 edition 3 – section – 11.9.4

> (Greater than) (Operator/relational)

Compares two operands to determine which is nearer to `+Infinity`.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`
See also:	Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), `NOT` Equal to (`!=`), NOT Identically equal to (`!==`), Relational operator

Cross-references:

ECMA 262 edition 2 – section – 11.8.2

ECMA 262 edition 3 – section – 11.8.2

>= (Greater than or equal to) (Operator/relational)

Compares two operands to determine which is nearer to `+Infinity` or whether they are equal

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`
See also:	Equal to (==), Greater than (>), Greater than or equal to (>=), Identically equal to (===), Less than (<), Less than or equal to (<=), NOT Equal to (`!=`), NOT Identically equal to (`!==`), Relational operator

Cross-references:

ECMA 262 edition 2 – section – 11.8.4

ECMA 262 edition 3 – section – 11.8.4

>> (Bitwise shift right) (Operator/bitwise)

Bitwise rightward shifts one operand according to another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.7.2

ECMA 262 edition 3 – section – 11.7.2

>>= (Bitwise shift right assign) (Operator/assignment)

Destructively bitwise rightward shifts the first of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Assignment operator, Bitwise operator, Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), LValue

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

>>> (Bitwise unsigned shift right) (Operator/bitwise)

Bitwise rightward shifts one operand according to another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server `version` – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`

See also:	Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), Shift operator

Cross-references:

ECMA 262 edition 2 – section – 11.7.3

ECMA 262 edition 3 – section – 11.7.3

>>>= (Bitwise unsigned shift right assign) (Operator/assignment)

Destructively bitwise rightward shifts the first of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Assignment operator, Bitwise operator, Bitwise shift left (<<), Bitwise shift left then assign (<<=), Bitwise shift operator, Bitwise shift right (>>), Bitwise shift right and assign (>>=), Bitwise unsigned shift right (>>>), Bitwise unsigned shift right and assign (>>>=), `LValue`

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

?: (Conditional block) (Operator/conditional)

Conditionally executes one code branch or another. Otherwise known as the Ternary operator.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Depends on arguments`
See also:	Conditionally execute (`?:`), `if( ... ) ..., if( ... ) ... else ...`

Cross-references:

ECMA 262 edition 2 – section – 11.12

ECMA 262 edition 3 – section – 11.12

@*/ (Pre-processor)

The closing pre-processor directive comment delimiter.

Availability:	JScript – 3.0 Internet Explorer – 4.0	
JavaScript syntax:	IE	`/*@someDirectives@*/`
Argument list:	`someDirectives`	One or more directives

See also:	Pre-processing, Pre-processing – `/*@ ... @*/`

@<variable_name> (Pre-processor)

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`User defined`

Refer to:

Pre-processing – `@<variable_name>`

@_alpha (Pre-processor)

A pre-processor constant indicating whether the script is running in a DEC alpha workstation.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_alpha`

@_jscript (Pre-processor)

A pre-processor constant indicating whether the script is executing in a JScript interpreter.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_jscript`

@_jscript_build (Pre-processor)

A pre-processor constant indicating the build version of the JScript environment.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Number primitive`

Refer to:

Pre-processing – `@_jscript_build`

@_jscript_version (Pre-processor)

A pre-processor constant indicating the version number of the JScript interpreter.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Number primitive`

Refer to:

Pre-processing – `@_jscript_version`

@_mac (Pre-processor)

A pre-processor constant indicating whether the script is running in a Macintosh workstation.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_mac`

@_mc680x0 (Pre-processor)

A pre-processor constant indicating whether the system contains a Motorola 68000 CPU.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_mc680x0`

@_PowerPC (Pre-processor)

A pre-processor constant indicating whether the system contains a Motorola PowerPC CPU.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_PowerPC`

@_win16 (Pre-processor)

A pre-processor constant indicating whether the script is running in a 16 bit Windows environment.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_win16`

@_win32 (Pre-processor)

A pre-processor constant indicating whether the script is running in a 32 bit Windows environment.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_win32`

@_x86 (Pre-processor)

A pre-processor constant indicating whether the system contains an Intel X-86 series CPU.

Availability:	JScript – 3.0 Internet Explorer – 4.0
Property/method value type:	`Boolean primitive`

Refer to:

Pre-processing – `@_x86`

@cc_on (Pre-processor)

A switch to activate the pre-processor phase of the script interpreter.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Pre-processing – `@cc_on`

@elif(...) ... (Pre-processor)

An optional `else-if` pre-processor token.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Pre-processing – `@elif( ... ) ...`

@else ... (Pre-processor)

Part of the conditional code use directive.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Pre-processing – `@else ...`

@end (Pre-processor)

Terminator for a conditional code block.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Pre-processing – `@end`

@if(...) ... (Pre-processor)

Conditionally includes a block of code.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Pre-processing – `@if( ... ) ...`

@set (Pre-processor)

Sets the contents of a pre-processor variable.

Availability:	JScript – 3.0 Internet Explorer – 4.0

Refer to:

Pre-processing – `@set`

[ ] (Array index) (Delimiter)

Array index delimiting tokens.

Availability:	ECMAScript edition – 2 JavaScript – 1.1 JScript – 3.0 Internet Explorer – 4.0 Netscape – 3.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Depends on array content`
See also:	Array index delimiter (`[ ]`), Property accessor

Cross-references:

ECMA 262 edition 2 – section – 7.6

ECMA 262 edition 2 – section – 11.2

ECMA 262 edition 3 – section – 7.7

[ ] (Property accessor) (Delimiter)

Properties can be accessed as if they were elements in an array.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0
See also:	Property accessor, Property name

Cross-references:

ECMA 262 edition 2 – section – 8.6.1

ECMA 262 edition 3 – section – 8.6.1

\ (Backslash) (Delimiter)

A means of escaping a character in quoted strings and regular expressions.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server– 2.0 Opera – 3.0

Refer to:

Escape sequence (\)

Cross-references:

ECMA 262 edition 2 – section – 2

ECMA 262 edition 2 – section – 6

ECMA 262 edition 2 – section – 7.7.3

ECMA 262 edition 3 – section – 2

ECMA 262 edition 3 – section – 6

ECMA 262 edition 3 – section – 7.7

^ (Bitwise XOR) (Operator/bitwise)

Bitwises XOR one operand with another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Binary bitwise operator, Bitwise XOR (^)

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.10

^= (Bitwise XOR assign) (Operator/assignment)

Bitwises XOR two operands and stores the result in the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Assignment operator, Bitwise operator, Bitwise XOR and assign (`^=`), `LValue`

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

_ (Underscore) (Symbol)

A special character allowed to be used in identifier names.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

Refer to:

Identifier

Cross-references:

ECMA 262 edition 2 – section – 7.5

ECMA 262 edition 3 – section – 7.6

__parent__ (Property)

A Netscape special scope chain inheritance property.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`ScopeChain` object	
JavaScript syntax:	N	`myClosure.__parent__`

With the `__parent__` property in Netscape, you can explicitly initialize the scope chain in your own custom functions.

Using this facility is somewhat inelegant, and since it is not portable its use is likely to be fairly restricted.

Warnings:

- It is unclear whether this property will be added to the ECMA standard. For now it is only available in Netscape.

Example code:

```
// Create an objectvar myObject =
{ aaa:"Some text" };// A fragment
of script to place that object into
the scope chainmyObject.__parent__ =
 this.__parent__;this.__parent__ = myObject;
```

See also:	`__proto__`, Lexical scoping, Scope, Scope chain

__proto__ (Property)

A Netscape special prototype inheritance property.

Availability:	JavaScript – 1.2 Netscape – 4.0	
Property/method value type:	`Function` object	
JavaScript syntax:	N	*myClosure*`.__proto__`

With the `__proto__` property in Netscape, you can explicitly initialize the prototype inheritance chain in your own custom constructors.

Warnings:

- It is unclear whether this property will be added to the ECMA standard. For now it is only available in Netscape.

Example code:

```
// Create a cuboid objectvar AAA = { length:100,
width:200, height:300 };// Create a cuboid that
shares the height valuevar BBB = { length:50,
width:100, __proto__:AAA };
```

See also:	`__parent__`, Subclasses

` (Backquote) (External code call)

Calls some external code during server side execution.

Refer to:

Backquote (`)

{ } (Braces) (Delimiter)

A delimiting token for a block of executable script source text.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0

See also:	`if( ... ) ...`,`if( ... ) ... else ...`, Code block delimiter {}

Cross-references:

ECMA 262 edition 2 – section – 12.5

ECMA 262 edition 3 – section – 12.1

| (Bitwise OR) (Operator/bitwise)

Bitwises `OR` one operand with another.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Binary bitwise operator, Bitwise OR (`|`)

Cross-references:

ECMA 262 edition 2 – section – 11.10

ECMA 262 edition 3 – section – 11.10

|= (Bitwise OR assign) (Operator/assignment)

Bitwises OR two operands storing the result in the first.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Assignment operator, Bitwise operator, Bitwise `OR` then assign (`|=`), `LValue`

Cross-references:

ECMA 262 edition 2 – section – 11.13

ECMA 262 edition 3 – section – 11.13

|| (Logical OR) (Operator/logical)

Logical `OR` of two operands.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server – 2.0 Opera – 3.0
Property/method value type:	`Boolean primitive`
See also:	Binary logical operator, Logical OR (||)

Cross-references:

ECMA 262 edition 2 – section – 11.11

ECMA 262 edition 3 – section – 11.11

~ (Bitwise NOT) (Operator/bitwise)

Bitwise `NOT` of an operand.

Availability:	ECMAScript edition – 2 JavaScript – 1.0 JScript – 1.0 Internet Explorer – 3.02 Netscape – 2.0 Netscape Enterprise Server– 2.0 Opera – 3.0
Property/method value type:	`Number primitive`
See also:	Bitwise NOT – complement (~), Bitwise operator, Logical NOT – complement (!), Unary expression, Unary operator

Cross-references:

ECMA 262 edition 2 – section – 11.4.8

ECMA 262 edition 3 – section – 11.4.8

! object (Object/HTML)

An object representing a `<!DOCTYPE>` DTD statement tag at the front of a document.

Availability:	DOM level – 1 JScript – 5.0 Internet Explorer – 5.0	
Inherits from:	`Element` object	
JavaScript syntax:	IE	*myDoctype* = *myDocument*.all.tags("!")[0]
	IE	*myDoctype* = *myDocument*.all[*anIndex*]
	IE	*myDoctype* = *myDocument*.getElementById (*anElementID*)
	IE	*myDoctype* = *myDocument*.getElementsByName(*aName*) [*anIndex*]
	IE	*myDoctype* = *myDocument*.doctype
	IE	*myDoctype* = *myDocument*.getElementsByTagName("!") [*anIndex*]
HTML syntax:	`<!DOCTYPE aDocumentDescription>`	
Argument list:	*aDocumentDescription*	A reference to a DTD for this document
	anIndex	A reference to an element in a collection
	aName	An associative array reference
	anElementID	The `ID` value of an `Element` object
Object properties:	`accessKey, tabIndex`	
Event handlers:	`onClick, onDblClick, onHelp, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp`	

The MSIE implementation creates an object that has a constructor name that suggests its class is "!". This is very odd, and if you intend to do any work on object classes, then naming a class with what might be interpreted as a special character in the script source may lead to some problems.

This is really a `Doctype` object and is specified in the DOM standard as a `<!DOCTYPE>` element.

The recommended means of access is to retrieve the value of the `doctype` property of the document you want to operate on. This is normally `document.doctype` for the current window but in the case of multiple frames, layers or windows you may be referring to the doctype of a different `document` object.

This object appears to inherit all of the properties of an HTML element.

See also:	`!.tabIndex`, `Document.doctype`, `Element` object, `Input.accessKey`

Property	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
accessKey	-	5.0 +	-	5.0 +	-	1 +	-	-
tabIndex	-	5.0 +	-	5.0 +	-	-	-	Warning

Event name	JavaScript	JScript	N	IE	Opera	DOM	HTML	Notes
onClick	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onDblClick	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onHelp	-	5.0 +	-	5.0 +	-	-	-	Warning
onKeyDown	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onKeyPress	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onKeyUp	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onMouseDown	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onMouseMove	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onMouseOut	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onMouseOver	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning
onMouseUp	-	5.0 +	-	5.0 +	-	-	4.0 +	Warning

Inheritance chain:

`Element` object, `Node` object

!.tabIndex (Property)

A tab index for a DTD statement.

Availability:	JScript – 5.0 Internet Explorer – 5.0	
Property/method value type:	`Number primitive`	
JavaScript syntax:	IE	*myDTD*`.tabIndex`

You'll likely only need to access properties of this object if you are writing 'extreme' JavaScript code. Most script writers will never encounter this object.

Warnings:

- This property is probably a mistake. It appeared when enumerating the properties of this object but the DTD contains no sensible tab accessible content nor is it part of a `<FORM>`. Under tests with normal DTD values this returned a zero.

See also:	`!` object, `Document.doctype`

Cross Reference by Entry Type

Each entry in this reference has a type or class, for example, 'object/HTML' for the A object. This cross-reference is ordered by entry type, with corresponding entries listed alphabetically beneath.

The collections, events, methods and properties entry types are slightly different because they list the collection, event, method or property name first and the actual entry name (which incorporates the object name) second.

Collections, events, methods and properties often have their own detailed entries, but information on browser support for them is also provided in the entry for the object to which they belong. References to both entries are listed where they exist.

An asterisk is used to show which entries are in the book. Entries without an asterisk will be found on the CD only.

Client pull techniques
Client-side JavaScript
Code signing
Collation sequence
Color names
Comma expression
Comment
Compatibility
Completion type
Compound statement
Conditional expression
Conditional operator
Conformance
Constant
Constant expression
Constraint
Constructor function
constructor property
constructor.name
Content Model
Control character
Conversion
Conversion to a Boolean
Conversion to a number
Conversion to a string
Conversion to an object
Core JavaScript
Core Object
Cross browser compatibility
Cross platform compatibility
Currency symbol
Custom object
Data Type
Date and time
Date constant
Daylight savings time adjustment
Debugger
Debugging - server side
Decimal value
Declaration
Declared function
Deep copying
Definition
Desktop JavaScript
Developing JavaScript source code
DHTML Behavior
Diagnostic message
Dialog boxes
Digit
Document component
Document event handlers
Domain error
Double-precision
Dynamic HTML
Dynamic positioning
Embedded JavaScript
Enquiry functions
Enumeration constant
Environment
Equality expression
Equality operator
Error
Error events
Error handling
Escape sequence (\)
Eval code
Event
Event bubbling
Event handler
Event handler in <SCRIPT>
Event handler properties
Event handler scope
Event management
Event model
Event names
Event propagation
Event-driven model
Exception
Exception handling
Executable code
Execute a function
Execution context
Execution environment
Exponent-log function
Expression
Expression statement
File extensions
Filter
Floating constant
Floating point
Floating point arithmetic
Floating point constant
Flow control
Form
Form element
Form verification
Formal Parameter List
Free-format language
Function
Function arguments
Function call
Function call operator ( )
Function code
Function definition
Function literal
Function object properties
Function property
Function prototype
Function scope
Fundamental data type
Furniture
Garbage collection
Global code
Global special variable
Glue code
Gotcha
Handler
Hexadecimal value
Hierarchy of objects
High order bit
Host environment
Host features
Host object
HTC
HTML Character entity
HTML Component
HTML file
HTML tag attribute
HyperLink
Identifier
Identifier resolution
Identity operator
Implementation
Implementation-defined behavior

Definition (continued)

R.E.
Range error
Raw event
Reference
Reference counting
Regex
RegExp literal
RegExp pattern
RegExp pattern - alternation
RegExp pattern - attributes
RegExp pattern - character class
RegExp pattern - character literal
RegExp pattern - extension syntax
RegExp pattern - grouping
RegExp pattern - position
RegExp pattern - references
RegExp pattern - repetition
RegExp pattern - sub-patterns
Regular expression*
Relational expression*
Relational operator*
Request-response loop
Reserved Word
Restricted access
rows object
RValue
Scalar type
Scope
Scope chain
Scope of event handler
Script
Script execution
Script fragment
Script Source Text
Script termination
Scriptlet
Security policy
Selection statement
Semantic event
Server-side JavaScript
Shallow copying
Shared Property
Shell Scripting with JavaScript
Shift expression
Shift operator
Side effect
Single line comment
Sort ordering
Source files
Source text
Special number values
Special type
SSJS
Standalone JavaScript
Statement
Static filters
Static method
Status line
Storage duration
String operator
Style sheet
Subclasses
Superclasses
Ternary operator
Thousands separator
Time range
Timeout handlers
Timer events
Token
Transition
Translation
Trigonometric function
TV Set-top boxes
Type
Type conversion
UDI
Unary expression
Unary operator
Undefined behavior
Undocumented features
Unspecified behavior
URI
URI handling functions
URL
URN
User-generated object
Utility objects
Value of an expression
Value preserving
Value property
Variable
Variable Declaration
Variable instantiation
Variable name
Variable statement
Visual filters
void expression
Web browser
Web scripting
Web server
Whitespace
Window adornments
Window events
Window feature list
Window furniture
XML name
Zero value

Delimiter

(Space)
(Single quote)
" (Double quote)
( ) (Argument delimiter)
( ) (Grouping operator)
*/ (Close comment block)
. (Decimal point)
. (Period)
/ (Slash)
/* ... */ (Comment block)
/*@ ... @*/ (Pre processing block)
// (Comment line)
: (Colon)
; (Semicolon)
[ ] (Array index)
[ ] (Property accessor)
\ (Backslash)
{ } (Braces)
Array index delimiter ([ ])*
Braces { }
Code block delimiter {}*
Colon (:)*
Comma operator (
Comment (// and /* ... */)*
Decimal point (.)*
Grouping operator ( )*

Delimiter (continued)

Object property delimiter (.)*
Parentheses ( )*
Pre-processing - /*@ ... @*/*
Quotation mark (" and ')
Semi-colon (;)*

Environment variable

CLASSPATH

Escape sequence

Newline*

Event handler

on ...
onAbort
onAbort, Image object*
onAbort, IMG object*
onAfterPrint
onAfterPrint, Dialog object*
onAfterPrint, Frame object*
onAfterPrint, Global object*
onAfterPrint, Window object*
onAfterUpdate
onAfterUpdate, Applet object*
onAfterUpdate, Area object*
onAfterUpdate, BODY object*
onAfterUpdate, Button object*
onAfterUpdate, BUTTON object*
onAfterUpdate, CAPTION object*
onAfterUpdate, Checkbox object*
onAfterUpdate, DIV object*
onAfterUpdate, Document object*
onAfterUpdate, FIELDSET object*
onAfterUpdate, FileUpload object*
onAfterUpdate, FormElement object*
onAfterUpdate, Hidden object*
onAfterUpdate, IMG object*
onAfterUpdate, Input object*
onAfterUpdate, MARQUEE object*
onAfterUpdate, OBJECT object*
onAfterUpdate, Password object*
onAfterUpdate, RadioButton object*
onAfterUpdate, ResetButton object*
onAfterUpdate, Select object*
onAfterUpdate, SubmitButton object*
onAfterUpdate, TABLE object*
onAfterUpdate, TD object*
onAfterUpdate, TEXTAREA object*
onAfterUpdate, TextCell object*
onAfterUpdate, TH object*
onBack
onBeforeCopy
onBeforeCut
onBeforeCut, Document object*
onBeforeEditFocus
onBeforeEditFocus, Document object*
onBeforePaste
onBeforePaste, Document object*
onBeforePrint
onBeforePrint, Dialog object*
onBeforePrint, Frame object*
onBeforePrint, Global object*
onBeforePrint, Window object*
onBeforeUnload
onBeforeUnload, BODY object*
onBeforeUnload, Dialog object*
onBeforeUnload, Frame object*
onBeforeUnload, FRAMESET object*
onBeforeUnload, Global object*
onBeforeUnload, TEXTAREA object*
onBeforeUnload, TH object*
onBeforeUnload, Window object*
onBeforeUpdate
onBeforeUpdate, Applet object*
onBeforeUpdate, Area object*
onBeforeUpdate, BODY object*
onBeforeUpdate, Button object*
onBeforeUpdate, BUTTON object*
onBeforeUpdate, CAPTION object*
onBeforeUpdate, Checkbox object*
onBeforeUpdate, DIV object*
onBeforeUpdate, Document object*
onBeforeUpdate, FIELDSET object*
onBeforeUpdate, FileUpload object*
onBeforeUpdate, FormElement object*
onBeforeUpdate, Hidden object*
onBeforeUpdate, IMG object*
onBeforeUpdate, Input object*
onBeforeUpdate, OBJECT object*
onBeforeUpdate, Password object*
onBeforeUpdate, RadioButton object*
onBeforeUpdate, ResetButton object*
onBeforeUpdate, Select object*
onBeforeUpdate, SubmitButton object*
onBeforeUpdate, TABLE object*
onBeforeUpdate, TD object*
onBeforeUpdate, TEXTAREA object*
onBeforeUpdate, TextCell object*
onBlur
onBlur, A object
onBlur, Applet object*
onBlur, Area object*
onBlur, Button object*
onBlur, BUTTON object*
onBlur, CAPTION object*
onBlur, Checkbox object*
onBlur, Dialog object*
onBlur, DIV object*
onBlur, Embed object*
onBlur, FIELDSET object*
onBlur, File object*
onBlur, FileUpload object*
onBlur, FormElement object*
onBlur, Frame object*
onBlur, Global object*
onBlur, Image object*
onBlur, IMG object*
onBlur, Input object*
onBlur, Label object*
onBlur, Layer object*
onBlur, Legend object*
onBlur, MARQUEE object*
onBlur, OBJECT object*
onBlur, Password object*
onBlur, RadioButton object*
onBlur, ResetButton object*
onBlur, Select object*
onBlur, SPAN object*
onBlur, SubmitButton object*
onBlur, TABLE object*
onBlur, TD object*

onBlur, TEXTAREA object*
onBlur, TextCell object*
onBlur, TH object*
onBlur, TR object*
onBlur, Window object*
onBounce
onBounce, MARQUEE object*
onChange
onChange, BODY object*
onChange, CAPTION object*
onChange, DIV object*
onChange, FIELDSET object*
onChange, FileUpload object*
onChange, FormElement object*
onChange, IMG object*
onChange, Input object*
onChange, Legend object*
onChange, Password object*
onChange, Select object*
onChange, TEXTAREA object*
onChange, TextCell object*
onClick
onClick, ! object*
onClick, A object
onClick, ABBR object
onClick, ACRONYM object
onClick, ADDRESS object
onClick, Anchor object*
onClick, Applet object*
onClick, Area object*
onClick, B object
onClick, BASE object*
onClick, BASEFONT object*
onClick, BDO object*
onClick, BGSOUND object*
onClick, BIG object
onClick, BLOCKQUOTE object*
onClick, BODY object*
onClick, BR object*
onClick, Button object*
onClick, BUTTON object*
onClick, CAPTION object*
onClick, CENTER object
onClick, Checkbox object*
onClick, CITE object
onClick, CODE object
onClick, COL object*
onClick, COLGROUP object*
onClick, DD object*
onClick, DEL object*
onClick, DFN object
onClick, DIR object*
onClick, DIV object*
onClick, DL object*
onClick, Document object*
onClick, DT object*
onClick, Element object*
onClick, EM object*
onClick, Embed object*
onClick, FIELDSET object*
onClick, File object*
onClick, FONT object*
onClick, Form object*
onClick, FormElement object*
onClick, FRAMESET object*
onClick, HEAD object*
onClick, H<n> object*
onClick, HR object*
onClick, HTML object*
onClick, HyperLink object*
onClick, I object*
onClick, IFRAME object*
onClick, Image object*
onClick, IMG object*
onClick, Input object*
onClick, INS object*
onClick, ISINDEX object*
onClick, KBD object*
onClick, Label object*
onClick, Legend object*
onClick, LI object*
onClick, LINK object*
onClick, LISTING object*
onClick, Location object*
onClick, Map object*
onClick, MARQUEE object*
onClick, MENU object*
onClick, META object*
onClick, NOFRAMES object*
onClick, NOSCRIPT object*
onClick, OBJECT object*
onClick, OL object*
onClick, OptGroupElement object*
onClick, Option object*
onClick, P object*
onClick, ParamElement object*
onClick, PLAINTEXT object
onClick, PRE object*
onClick, Q object
onClick, RadioButton object*
onClick, ResetButton object*
onClick, S object
onClick, SAMP object
onClick, SCRIPT object*
onClick, SMALL object
onClick, SPAN object*
onClick, STRIKE object
onClick, STRONG object*
onClick, STYLE object (1)*
onClick, style object (2)*
onClick, SUB object
onClick, SubmitButton object*
onClick, SUP object
onClick, TABLE object*
onClick, TableColElement object*
onClick, TBODY object*
onClick, TD object*
onClick, TextRange object*
onClick, TFOOT object*
onClick, TH object*
onClick, THEAD object*
onClick, TITLE object*
onClick, TR object*
onClick, TT object*
onClick, U object*
onClick, UL object*
onClick, Url object*
onClick, VAR object
onContentReady
onContextMenu
onContextMenu, Document object*
onCopy
onCut
onCut, Document object*

Event handler (continued)

onDataAvailable
onDataAvailable, Applet object*
onDataAvailable, Area object*
onDataAvailable, BODY object*
onDataAvailable, IMG object*
onDataAvailable, OBJECT object*
onDataAvailable, XML object*
onDataSetChanged
onDataSetChanged, Applet object*
onDataSetChanged, Area object*
onDataSetChanged, BODY object*
onDataSetChanged, IMG object*
onDatasetChanged, OBJECT object*
onDataSetChanged, XML object*
onDataSetComplete
onDataSetComplete, Applet object*
onDataSetComplete, Area object*
onDataSetComplete, BODY object*
onDataSetComplete, IMG object*
onDatasetComplete, OBJECT object*
onDataSetComplete, XML object*
onDblClick
onDblClick, ! object*
onDblClick, A object
onDblClick, ABBR object
onDblClick, ACRONYM object
onDblClick, ADDRESS object
onDblClick, Applet object*
onDblClick, Area object*
onDblClick, B object
onDblClick, BASE object*
onDblClick, BASEFONT object*
onDblClick, BDO object*
onDblClick, BGSOUND object*
onDblClick, BIG object
onDblClick, BLOCKQUOTE object*
onDblClick, BODY object*
onDblClick, BR object*
onDblClick, Button object*
onDblClick, BUTTON object*
onDblClick, CAPTION object*
onDblClick, CENTER object
onDblClick, Checkbox object*
onDblClick, CITE object
onDblClick, CODE object
onDblClick, COL object*
onDblClick, COLGROUP object*
onDblClick, DD object*
onDblClick, DEL object*
onDblClick, DFN object
onDblClick, DIR object*
onDblClick, DIV object*
onDblClick, DL object*
onDblClick, Document object*
onDblClick, DT object*
onDblClick, Element object*
onDblClick, EM object*
onDblClick, Embed object*
onDblClick, FIELDSET object*
onDblClick, File object*
onDblClick, FONT object*
onDblClick, Form object*
onDblClick, FormElement object*
onDblClick, FRAMESET object*
onDblClick, HEAD object*
onDblClick, H<n> object*
onDblClick, HR object*
onDblClick, HTML object*
onDblClick, HyperLink object*
onDblClick, I object*
onDblClick, IFRAME object*
onDblClick, Image object*
onDblClick, IMG object*
onDblClick, Input object*
onDblClick, INS object*
onDblClick, ISINDEX object*
onDblClick, KBD object*
onDblClick, Label object*
onDblClick, Legend object*
onDblClick, LI object*
onDblClick, LINK object*
onDblClick, LISTING object*
onDblClick, Location object*
onDblClick, Map object*
onDblClick, MARQUEE object*
onDblClick, MENU object*
onDblClick, META object*
onDblClick, NOFRAMES object*
onDblClick, NOSCRIPT object*
onDblClick, OBJECT object*
onDblClick, OL object*
onDblClick, OptGroupElement object*
onDblClick, Option object*
onDblClick, P object*
onDblClick, ParamElement object*
onDblClick, PLAINTEXT object
onDblClick, PRE object*
onDblClick, Q object
onDblClick, RadioButton object*
onDblClick, ResetButton object*
onDblClick, S object
onDblClick, SAMP object
onDblClick, SCRIPT object*
onDblClick, SMALL object
onDblClick, SPAN object*
onDblClick, STRIKE object
onDblClick, STRONG object*
onDblClick, STYLE object (1)*
onDblClick, style object (2)*
onDblClick, SUB object
onDblClick, SubmitButton object*
onDblClick, SUP object
onDblClick, TABLE object*
onDblClick, TableColElement object*
onDblClick, TBODY object*
onDblClick, TD object*
onDblClick, TextRange object*
onDblClick, TFOOT object*
onDblClick, TH object*
onDblClick, THEAD object*
onDblClick, TITLE object*
onDblClick, TR object*
onDblClick, TT object*
onDblClick, U object*
onDblClick, UL object*
onDblClick, Url object*
onDblClick, VAR object
onDocumentReady
onDrag
onDrag, Document object*
onDragDrop
onDragDrop, Dialog object*
onDragDrop, Frame object*

onDragDrop, Global object*
onDragDrop, Window object*
onDragEnd
onDragEnd, Document object*
onDragEnter
onDragEnter, Document object*
onDragLeave
onDragLeave, Document object*
onDragOver
onDragOver, Document object*
onDragStart
onDragStart, ACRONYM object
onDragStart, ADDRESS object
onDragStart, B object
onDragStart, BIG object
onDragStart, BLOCKQUOTE object*
onDragStart, BODY object*
onDragStart, BUTTON object*
onDragStart, CAPTION object*
onDragStart, CENTER object
onDragStart, CITE object
onDragStart, CODE object
onDragStart, DD object*
onDragStart, DEL object*
onDragStart, DFN object
onDragStart, DIR object*
onDragStart, DIV object*
onDragStart, DL object*
onDragStart, Document object*
onDragStart, DT object*
onDragStart, EM object*
onDragStart, FIELDSET object*
onDragStart, FileUpload object*
onDragStart, FONT object*
onDragStart, Form object*
onDragStart, H<n> object*
onDragStart, HR object*
onDragStart, I object*
onDragStart, IMG object*
onDragStart, INS object*
onDragStart, KBD object*
onDragStart, Label object*
onDragStart, Legend object*
onDragStart, LI object*
onDragStart, LISTING object*
onDragStart, MARQUEE object*
onDragStart, MENU object*
onDragStart, OBJECT object*
onDragStart, OL object*
onDragStart, P object*
onDragStart, PLAINTEXT object
onDragStart, PRE object*
onDragStart, Q object
onDragStart, S object
onDragStart, SAMP object
onDragStart, Select object*
onDragStart, SMALL object
onDragStart, SPAN object*
onDragStart, STRIKE object
onDragStart, STRONG object*
onDragStart, SUB object
onDragStart, SUP object
onDragStart, TABLE object*
onDragStart, TBODY object*
onDragStart, TD object*
onDragStart, TEXTAREA object*
onDragStart, TFOOT object*
onDragStart, TH object*
onDragStart, THEAD object*
onDragStart, TR object*
onDragStart, TT object*
onDragStart, U object*
onDragStart, UL object*
onDragStart, VAR object
onDrop
onDrop, Document object*
onError
onError, Dialog object*
onError, Frame object*
onError, Global object*
onError, Image object*
onError, IMG object*
onError, LINK object*
onError, OBJECT object*
onError, SCRIPT object*
onError, STYLE object (1)*
onError, Window object*
onErrorUpdate
onErrorUpdate, Applet object*
onErrorUpdate, Area object*
onErrorUpdate, BODY object*
onErrorUpdate, Button object*
onErrorUpdate, CAPTION object*
onErrorUpdate, Checkbox object*
onErrorUpdate, Document object*
onErrorUpdate, FIELDSET object*
onErrorUpdate, OBJECT object*
onErrorUpdate, RadioButton object*
onErrorUpdate, TEXTAREA object*
onFilterChange
onFilterChange, ACRONYM object
onFilterChange, ADDRESS object
onFilterChange, B object
onFilterChange, BIG object
onFilterChange, BLOCKQUOTE object*
onFilterChange, BODY object*
onFilterChange, Button object*
onFilterChange, BUTTON object*
onFilterChange, CAPTION object*
onFilterChange, CENTER object
onFilterChange, Checkbox object*
onFilterChange, CITE object
onFilterChange, CODE object
onFilterChange, DD object*
onFilterChange, DEL object*
onFilterChange, DFN object
onFilterChange, DIR object*
onFilterChange, DL object*
onFilterChange, DT object*
onFilterChange, EM object*
onFilterChange, FIELDSET object*
onFilterChange, FileUpload object*
onFilterChange, FONT object*
onFilterChange, Form object*
onFilterChange, H<n> object*
onFilterChange, HR object*
onFilterChange, I object*
onFilterChange, IMG object*
onFilterChange, INS object*
onFilterChange, KBD object*
onFilterChange, Label object*
onFilterChange, Legend object*
onFilterChange, LI object*
onFilterChange, LISTING object*

Event handler (continued)

onFilterChange, MARQUEE object*
onFilterChange, MENU object*
onFilterChange, OBJECT object*
onFilterChange, OL object*
onFilterChange, P object*
onFilterChange, Password object*
onFilterChange, PLAINTEXT object
onFilterChange, PRE object*
onFilterChange, Q object
onFilterChange, RadioButton object*
onFilterChange, ResetButton object*
onFilterChange, S object
onFilterChange, SAMP object
onFilterChange, Select object*
onFilterChange, SMALL object
onFilterChange, SPAN object*
onFilterChange, STRIKE object
onFilterChange, STRONG object*
onFilterChange, SUB object
onFilterChange, SubmitButton object*
onFilterChange, SUP object
onFilterChange, TABLE object*
onFilterChange, TBODY object*
onFilterChange, TD object*
onFilterChange, TEXTAREA object*
onFilterChange, TextCell object*
onFilterChange, TFOOT object*
onFilterChange, TH object*
onFilterChange, THEAD object*
onFilterChange, TR object*
onFilterChange, TT object*
onFilterChange, U object*
onFilterChange, UL object*
onFilterChange, VAR object
onFinish
onFinish, MARQUEE object*
onFocus
onFocus, A object
onFocus, Applet object*
onFocus, Area object*
onFocus, Button object*
onFocus, BUTTON object*
onFocus, CAPTION object*
onFocus, Checkbox object*
onFocus, Dialog object*
onFocus, DIV object*
onFocus, Embed object*
onFocus, FIELDSET object*
onFocus, File object*
onFocus, FileUpload object*
onFocus, FormElement object*
onFocus, Frame object*
onFocus, Global object*
onFocus, Image object*
onFocus, IMG object*
onFocus, Input object*
onFocus, Label object*
onFocus, Layer object*
onFocus, Legend object*
onFocus, MARQUEE object*
onFocus, OBJECT object*
onFocus, Password object*
onFocus, RadioButton object*
onFocus, ResetButton object*
onFocus, Select object*
onFocus, SubmitButton object*
onFocus, TABLE object*
onFocus, TEXTAREA object*
onFocus, TextCell object*
onFocus, Window object*
onForward
onHelp
onHelp, ! object*
onHelp, A object
onHelp, ABBR object
onHelp, ACRONYM object
onHelp, ADDRESS object
onHelp, Applet object*
onHelp, Area object*
onHelp, B object
onHelp, BASE object*
onHelp, BASEFONT object*
onHelp, BDO object*
onHelp, BGSOUND object*
onHelp, BIG object
onHelp, BLOCKQUOTE object*
onHelp, BODY object*
onHelp, BR object*
onHelp, Button object*
onHelp, BUTTON object*
onHelp, CAPTION object*
onHelp, CENTER object
onHelp, Checkbox object*
onHelp, CITE object
onHelp, CODE object
onHelp, COL object*
onHelp, COLGROUP object*
onHelp, DD object*
onHelp, DEL object*
onHelp, DFN object
onHelp, Dialog object*
onHelp, DIR object*
onHelp, DIV object*
onHelp, DL object*
onHelp, Document object*
onHelp, DT object*
onHelp, Element object*
onHelp, EM object*
onHelp, Embed object*
onHelp, FIELDSET object*
onHelp, FileUpload object*
onHelp, FONT object*
onHelp, Form object*
onHelp, FormElement object*
onHelp, Frame object*
onHelp, FRAMESET object*
onHelp, Global object*
onHelp, HEAD object*
onHelp, Hidden object*
onHelp, H<n> object*
onHelp, HR object*
onHelp, HTML object*
onHelp, HyperLink object*
onHelp, I object*
onHelp, IFRAME object*
onHelp, Image object*
onHelp, IMG object*
onHelp, Input object*
onHelp, INS object*
onHelp, ISINDEX object*
onHelp, KBD object*
onHelp, Label object*
onHelp, Legend object*

onHelp, LI object*
onHelp, LINK object*
onHelp, LISTING object*
onHelp, Location object*
onHelp, Map object*
onHelp, MARQUEE object*
onHelp, MENU object*
onHelp, META object*
onHelp, NOFRAMES object*
onHelp, NOSCRIPT object*
onHelp, OBJECT object*
onHelp, OL object*
onHelp, OptGroupElement object*
onHelp, Option object*
onHelp, P object*
onHelp, ParamElement object*
onHelp, Password object*
onHelp, PLAINTEXT object
onHelp, PRE object*
onHelp, Q object
onHelp, RadioButton object*
onHelp, ResetButton object*
onHelp, S object
onHelp, SAMP object
onHelp, SCRIPT object*
onHelp, Select object*
onHelp, SMALL object
onHelp, SPAN object*
onHelp, STRIKE object
onHelp, STRONG object*
onHelp, STYLE object (1)*
onHelp, style object (2)*
onHelp, SUB object
onHelp, SubmitButton object*
onHelp, SUP object
onHelp, TABLE object*
onHelp, TableColElement object*
onHelp, TBODY object*
onHelp, TD object*
onHelp, TEXTAREA object*
onHelp, TextCell object*
onHelp, TextRange object*
onHelp, TFOOT object*
onHelp, TH object*
onHelp, THEAD object*
onHelp, TITLE object*
onHelp, TR object*
onHelp, TT object*
onHelp, U object*
onHelp, UL object*
onHelp, Url object*
onHelp, VAR object
onHelp, Window object*
onKeyDown
onKeyDown, ! object*
onKeyDown, A object
onKeyDown, ABBR object
onKeyDown, ACRONYM object
onKeyDown, ADDRESS object
onKeyDown, Applet object*
onKeyDown, Area object*
onKeyDown, B object
onKeyDown, BASE object*
onKeyDown, BASEFONT object*
onKeyDown, BDO object*
onKeyDown, BGSOUND object*
onKeyDown, BIG object
onKeyDown, BLOCKQUOTE object*
onKeyDown, BODY object*
onKeyDown, BR object*
onKeyDown, Button object*
onKeyDown, BUTTON object*
onKeyDown, CAPTION object*
onKeyDown, CENTER object
onKeyDown, Checkbox object*
onKeyDown, CITE object
onKeyDown, CODE object
onKeyDown, COL object*
onKeyDown, COLGROUP object*
onKeyDown, DD object*
onKeyDown, DEL object*
onKeyDown, DFN object
onKeyDown, DIR object*
onKeyDown, DIV object*
onKeyDown, DL object*
onKeyDown, Document object*
onKeyDown, DT object*
onKeyDown, Element object*
onKeyDown, EM object*
onKeyDown, Embed object*
onKeyDown, FIELDSET object*
onKeyDown, FileUpload object*
onKeyDown, FONT object*
onKeyDown, Form object*
onKeyDown, FormElement object*
onKeyDown, FRAMESET object*
onKeyDown, HEAD object*
onKeyDown, H<n> object*
onKeyDown, HR object*
onKeyDown, HTML object*
onKeyDown, HyperLink object*
onKeyDown, I object*
onKeyDown, IFRAME object*
onKeyDown, Image object*
onKeyDown, IMG object*
onKeyDown, Input object*
onKeyDown, INS object*
onKeyDown, ISINDEX object*
onKeyDown, KBD object*
onKeyDown, Label object*
onKeyDown, Legend object*
onKeyDown, LI object*
onKeyDown, LINK object*
onKeyDown, LISTING object*
onKeyDown, Location object*
onKeyDown, Map object*
onKeyDown, MARQUEE object*
onKeyDown, MENU object*
onKeyDown, META object*
onKeyDown, NOFRAMES object*
onKeyDown, NOSCRIPT object*
onKeyDown, OBJECT object*
onKeyDown, OL object*
onKeyDown, OptGroupElement object*
onKeyDown, Option object*
onKeyDown, P object*
onKeyDown, ParamElement object*
onKeyDown, Password object*
onKeyDown, PLAINTEXT object
onKeyDown, PRE object*
onKeyDown, Q object
onKeyDown, RadioButton object*
onKeyDown, ResetButton object*
onKeyDown, S object

Event handler (continued)

onKeyDown, SAMP object
onKeyDown, SCRIPT object*
onKeyDown, Select object*
onKeyDown, SMALL object
onKeyDown, SPAN object*
onKeyDown, STRIKE object
onKeyDown, STRONG object*
onKeyDown, STYLE object (1)*
onKeyDown, style object (2)*
onKeyDown, SUB object
onKeyDown, SubmitButton object*
onKeyDown, SUP object
onKeyDown, TABLE object*
onKeyDown, TableColElement object*
onKeyDown, TBODY object*
onKeyDown, TD object*
onKeyDown, TEXTAREA object*
onKeyDown, TextCell object*
onKeyDown, TextRange object*
onKeyDown, TFOOT object*
onKeyDown, TH object*
onKeyDown, THEAD object*
onKeyDown, TITLE object*
onKeyDown, TR object*
onKeyDown, TT object*
onKeyDown, U object*
onKeyDown, UL object*
onKeyDown, Url object*
onKeyDown, VAR object
onKeyPress
onKeyPress, ! object*
onKeyPress, A object
onKeyPress, ABBR object
onKeyPress, ACRONYM object
onKeyPress, ADDRESS object
onKeyPress, Applet object*
onKeyPress, Area object*
onKeyPress, B object
onKeyPress, BASE object*
onKeyPress, BASEFONT object*
onKeyPress, BDO object*
onKeyPress, BGSOUND object*
onKeyPress, BIG object
onKeyPress, BLOCKQUOTE object*
onKeyPress, BODY object*
onKeyPress, BR object*
onKeyPress, Button object*
onKeyPress, BUTTON object*
onKeyPress, CAPTION object*
onKeyPress, CENTER object
onKeyPress, Checkbox object*
onKeyPress, CITE object
onKeyPress, CODE object
onKeyPress, COL object*
onKeyPress, COLGROUP object*
onKeyPress, DD object*
onKeyPress, DEL object*
onKeyPress, DFN object
onKeyPress, DIR object*
onKeyPress, DIV object*
onKeyPress, DL object*
onKeyPress, Document object*
onKeyPress, DT object*
onKeyPress, Element object*
onKeyPress, EM object*
onKeyPress, Embed object*
onKeyPress, FIELDSET object*
onKeyPress, FileUpload object*
onKeyPress, FONT object*
onKeyPress, Form object*
onKeyPress, FormElement object*
onKeyPress, FRAMESET object*
onKeyPress, HEAD object*
onKeyPress, H<n> object*
onKeyPress, HR object*
onKeyPress, HTML object*
onKeyPress, HyperLink object*
onKeyPress, I object*
onKeyPress, IFRAME object*
onKeyPress, Image object*
onKeyPress, IMG object*
onKeyPress, Input object*
onKeyPress, INS object*
onKeyPress, ISINDEX object*
onKeyPress, KBD object*
onKeyPress, Label object*
onKeyPress, Legend object*
onKeyPress, LI object*
onKeyPress, LINK object*
onKeyPress, LISTING object*
onKeyPress, Location object*
onKeyPress, Map object*
onKeyPress, MARQUEE object*
onKeyPress, MENU object*
onKeyPress, META object*
onKeyPress, NOFRAMES object*
onKeyPress, NOSCRIPT object*
onKeyPress, OBJECT object*
onKeyPress, OL object*
onKeyPress, OptGroupElement object*
onKeyPress, Option object*
onKeyPress, P object*
onKeyPress, ParamElement object*
onKeyPress, Password object*
onKeyPress, PLAINTEXT object
onKeyPress, PRE object*
onKeyPress, Q object
onKeyPress, RadioButton object*
onKeyPress, ResetButton object*
onKeyPress, S object
onKeyPress, SAMP object
onKeyPress, SCRIPT object*
onKeyPress, Select object*
onKeyPress, SMALL object
onKeyPress, SPAN object*
onKeyPress, STRIKE object
onKeyPress, STRONG object*
onKeyPress, STYLE object (1)*
onKeyPress, style object (2)*
onKeyPress, SUB object
onKeyPress, SubmitButton object*
onKeyPress, SUP object
onKeyPress, TABLE object*
onKeyPress, TableColElement object*
onKeyPress, TBODY object*
onKeyPress, TD object*
onKeyPress, TEXTAREA object*
onKeyPress, TextCell object*
onKeyPress, TextRange object*
onKeyPress, TFOOT object*
onKeyPress, TH object*
onKeyPress, THEAD object*
onKeyPress, TITLE object*

onKeyPress, TR object*
onKeyPress, TT object*
onKeyPress, U object*
onKeyPress, UL object*
onKeyPress, Url object*
onKeyPress, VAR object
onKeyUp
onKeyUp, ! object*
onKeyUp, A object
onKeyUp, ABBR object
onKeyUp, ACRONYM object
onKeyUp, ADDRESS object
onKeyUp, Applet object*
onKeyUp, Area object*
onKeyUp, B object
onKeyUp, BASE object*
onKeyUp, BASEFONT object*
onKeyUp, BDO object*
onKeyUp, BGSOUND object*
onKeyUp, BIG object
onKeyUp, BLOCKQUOTE object*
onKeyUp, BODY object*
onKeyUp, BR object*
onKeyUp, Button object*
onKeyUp, BUTTON object*
onKeyUp, CAPTION object*
onKeyUp, CENTER object
onKeyUp, Checkbox object*
onKeyUp, CITE object
onKeyUp, CODE object
onKeyUp, COL object*
onKeyUp, COLGROUP object*
onKeyUp, DD object*
onKeyUp, DEL object*
onKeyUp, DFN object
onKeyUp, DIR object*
onKeyUp, DIV object*
onKeyUp, DL object*
onKeyUp, Document object*
onKeyUp, DT object*
onKeyUp, Element object*
onKeyUp, EM object*
onKeyUp, Embed object*
onKeyUp, FIELDSET object*
onKeyUp, FileUpload object*
onKeyUp, FONT object*
onKeyUp, Form object*
onKeyUp, FormElement object*
onKeyUp, FRAMESET object*
onKeyUp, HEAD object*
onKeyUp, H<n> object*
onKeyUp, HR object*
onKeyUp, HTML object*
onKeyUp, HyperLink object*
onKeyUp, I object*
onKeyUp, IFRAME object*
onKeyUp, Image object*
onKeyUp, IMG object*
onKeyUp, Input object*
onKeyUp, INS object*
onKeyUp, ISINDEX object*
onKeyUp, KBD object*
onKeyUp, Label object*
onKeyUp, Legend object*
onKeyUp, LI object*
onKeyUp, LINK object*
onKeyUp, LISTING object*
onKeyUp, Location object*
onKeyUp, Map object*
onKeyUp, MARQUEE object*
onKeyUp, MENU object*
onKeyUp, META object*
onKeyUp, NOFRAMES object*
onKeyUp, NOSCRIPT object*
onKeyUp, OBJECT object*
onKeyUp, OL object*
onKeyUp, OptGroupElement object*
onKeyUp, Option object*
onKeyUp, P object*
onKeyUp, ParamElement object*
onKeyUp, Password object*
onKeyUp, PLAINTEXT object
onKeyUp, PRE object*
onKeyUp, Q object
onKeyUp, RadioButton object*
onKeyUp, ResetButton object*
onKeyUp, S object
onKeyUp, SAMP object
onKeyUp, SCRIPT object*
onKeyUp, Select object*
onKeyUp, SMALL object
onKeyUp, SPAN object*
onKeyUp, STRIKE object
onKeyUp, STRONG object*
onKeyUp, STYLE object (1)*
onKeyUp, style object (2)*
onKeyUp, SUB object
onKeyUp, SubmitButton object*
onKeyUp, SUP object
onKeyUp, TABLE object*
onKeyUp, TableColElement object*
onKeyUp, TBODY object*
onKeyUp, TD object*
onKeyUp, TEXTAREA object*
onKeyUp, TextCell object*
onKeyUp, TextRange object*
onKeyUp, TFOOT object*
onKeyUp, TH object*
onKeyUp, THEAD object*
onKeyUp, TITLE object*
onKeyUp, TR object*
onKeyUp, TT object*
onKeyUp, U object*
onKeyUp, UL object*
onKeyUp, Url object*
onKeyUp, VAR object
onLoad
onLoad, Applet object*
onLoad, Area object*
onLoad, Dialog object*
onLoad, Frame object*
onLoad, FRAMESET object*
onLoad, Global object*
onLoad, Image object*
onLoad, IMG object*
onLoad, Layer object*
onLoad, LINK object*
onLoad, SCRIPT object*
onLoad, STYLE object (1)*
onLoad, Window object*
onLoseCapture
onMouseDown
onMouseDown, ! object*
onMouseDown, A object

Event handler (continued)

onMouseDown, ABBR object
onMouseDown, ACRONYM object
onMouseDown, ADDRESS object
onMouseDown, Anchor object*
onMouseDown, Applet object*
onMouseDown, Area object*
onMouseDown, B object
onMouseDown, BASE object*
onMouseDown, BASEFONT object*
onMouseDown, BDO object*
onMouseDown, BGSOUND object*
onMouseDown, BIG object
onMouseDown, BLOCKQUOTE object*
onMouseDown, BODY object*
onMouseDown, BR object*
onMouseDown, Button object*
onMouseDown, BUTTON object*
onMouseDown, CAPTION object*
onMouseDown, CENTER object
onMouseDown, Checkbox object*
onMouseDown, CITE object
onMouseDown, CODE object
onMouseDown, COL object*
onMouseDown, COLGROUP object*
onMouseDown, DD object*
onMouseDown, DEL object*
onMouseDown, DFN object
onMouseDown, DIR object*
onMouseDown, DIV object*
onMouseDown, DL object*
onMouseDown, Document object*
onMouseDown, DT object*
onMouseDown, Element object*
onMouseDown, EM object*
onMouseDown, Embed object*
onMouseDown, FIELDSET object*
onMouseDown, FileUpload object*
onMouseDown, FONT object*
onMouseDown, Form object*
onMouseDown, FormElement object*
onMouseDown, FRAMESET object*
onMouseDown, HEAD object*
onMouseDown, H<n> object*
onMouseDown, HR object*
onMouseDown, HTML object*
onMouseDown, HyperLink object*
onMouseDown, I object*
onMouseDown, IFRAME object*
onMouseDown, Image object*
onMouseDown, IMG object*
onMouseDown, Input object*
onMouseDown, INS object*
onMouseDown, ISINDEX object*
onMouseDown, KBD object*
onMouseDown, Label object*
onMouseDown, Legend object*
onMouseDown, LI object*
onMouseDown, LINK object*
onMouseDown, LISTING object*
onMouseDown, Location object*
onMouseDown, Map object*
onMouseDown, MARQUEE object*
onMouseDown, MENU object*
onMouseDown, META object*
onMouseDown, NOFRAMES object*
onMouseDown, NOSCRIPT object*
onMouseDown, OBJECT object*
onMouseDown, OL object*
onMouseDown, OptGroupElement object*
onMouseDown, Option object*
onMouseDown, P object*
onMouseDown, ParamElement object*
onMouseDown, Password object*
onMouseDown, PLAINTEXT object
onMouseDown, PRE object*
onMouseDown, Q object
onMouseDown, RadioButton object*
onMouseDown, ResetButton object*
onMouseDown, S object
onMouseDown, SAMP object
onMouseDown, SCRIPT object*
onMouseDown, Select object*
onMouseDown, SMALL object
onMouseDown, SPAN object*
onMouseDown, STRIKE object
onMouseDown, STRONG object*
onMouseDown, STYLE object (1)*
onMouseDown, style object (2)*
onMouseDown, SUB object
onMouseDown, SubmitButton object*
onMouseDown, SUP object
onMouseDown, TABLE object*
onMouseDown, TableColElement object*
onMouseDown, TBODY object*
onMouseDown, TD object*
onMouseDown, TEXTAREA object*
onMouseDown, TextCell object*
onMouseDown, TextRange object*
onMouseDown, TFOOT object*
onMouseDown, TH object*
onMouseDown, THEAD object*
onMouseDown, TITLE object*
onMouseDown, TR object*
onMouseDown, TT object*
onMouseDown, U object*
onMouseDown, UL object*
onMouseDown, Url object*
onMouseDown, VAR object
onMouseDrag
onMouseMove
onMouseMove, ! object*
onMouseMove, A object
onMouseMove, ABBR object
onMouseMove, ACRONYM object
onMouseMove, ADDRESS object
onMouseMove, Applet object*
onMouseMove, Area object*
onMouseMove, B object
onMouseMove, BASE object*
onMouseMove, BASEFONT object*
onMouseMove, BDO object*
onMouseMove, BGSOUND object*
onMouseMove, BIG object
onMouseMove, BLOCKQUOTE object*
onMouseMove, BODY object*
onMouseMove, BR object*
onMouseMove, Button object*
onMouseMove, BUTTON object*
onMouseMove, CAPTION object*
onMouseMove, CENTER object
onMouseMove, Checkbox object*
onMouseMove, CITE object
onMouseMove, CODE object

onMouseMove, COL object*
onMouseMove, COLGROUP object*
onMouseMove, DD object*
onMouseMove, DEL object*
onMouseMove, DFN object
onMouseMove, Dialog object*
onMouseMove, DIR object*
onMouseMove, DIV object*
onMouseMove, DL object*
onMouseMove, Document object*
onMouseMove, DT object*
onMouseMove, Element object*
onMouseMove, EM object*
onMouseMove, Embed object*
onMouseMove, FIELDSET object*
onMouseMove, FileUpload object*
onMouseMove, FONT object*
onMouseMove, Form object*
onMouseMove, FormElement object*
onMouseMove, Frame object*
onMouseMove, FRAMESET object*
onMouseMove, HEAD object*
onMouseMove, H<n> object*
onMouseMove, HR object*
onMouseMove, HTML object*
onMouseMove, HyperLink object*
onMouseMove, I object*
onMouseMove, IFRAME object*
onMouseMove, Image object*
onMouseMove, IMG object*
onMouseMove, Input object*
onMouseMove, INS object*
onMouseMove, ISINDEX object*
onMouseMove, KBD object*
onMouseMove, Label object*
onMouseMove, Legend object*
onMouseMove, LI object*
onMouseMove, LINK object*
onMouseMove, LISTING object*
onMouseMove, Location object*
onMouseMove, Map object*
onMouseMove, MARQUEE object*
onMouseMove, MENU object*
onMouseMove, META object*
onMouseMove, NOFRAMES object*
onMouseMove, NOSCRIPT object*
onMouseMove, OBJECT object*
onMouseMove, OL object*
onMouseMove, OptGroupElement object*
onMouseMove, Option object*
onMouseMove, P object*
onMouseMove, ParamElement object*
onMouseMove, Password object*
onMouseMove, PLAINTEXT object
onMouseMove, PRE object*
onMouseMove, Q object
onMouseMove, RadioButton object*
onMouseMove, ResetButton object*
onMouseMove, S object
onMouseMove, SAMP object
onMouseMove, SCRIPT object*
onMouseMove, Select object*
onMouseMove, SMALL object
onMouseMove, SPAN object*
onMouseMove, STRIKE object
onMouseMove, STRONG object*
onMouseMove, STYLE object (1)*
onMouseMove, style object (2)*
onMouseMove, SUB object
onMouseMove, SubmitButton object*
onMouseMove, SUP object
onMouseMove, TABLE object*
onMouseMove, TableColElement object*
onMouseMove, TBODY object*
onMouseMove, TD object*
onMouseMove, TEXTAREA object*
onMouseMove, TextCell object*
onMouseMove, TextRange object*
onMouseMove, TFOOT object*
onMouseMove, TH object*
onMouseMove, THEAD object*
onMouseMove, TITLE object*
onMouseMove, TR object*
onMouseMove, TT object*
onMouseMove, U object*
onMouseMove, UL object*
onMouseMove, Url object*
onMouseMove, VAR object
onMouseMove, Window object*
onMouseOut
onMouseOut, ! object*
onMouseOut, A object
onMouseOut, ABBR object
onMouseOut, ACRONYM object
onMouseOut, ADDRESS object
onMouseOut, Anchor object*
onMouseOut, Applet object*
onMouseOut, Area object*
onMouseOut, B object
onMouseOut, BASE object*
onMouseOut, BASEFONT object*
onMouseOut, BDO object*
onMouseOut, BGSOUND object*
onMouseOut, BIG object
onMouseOut, BLOCKQUOTE object*
onMouseOut, BODY object*
onMouseOut, BR object*
onMouseOut, Button object*
onMouseOut, BUTTON object*
onMouseOut, CAPTION object*
onMouseOut, CENTER object
onMouseOut, Checkbox object*
onMouseOut, CITE object
onMouseOut, CODE object
onMouseOut, COL object*
onMouseOut, COLGROUP object*
onMouseOut, DD object*
onMouseOut, DEL object*
onMouseOut, DFN object
onMouseOut, DIR object*
onMouseOut, DIV object*
onMouseOut, DL object*
onMouseOut, Document object*
onMouseOut, DT object*
onMouseOut, Element object*
onMouseOut, EM object*
onMouseOut, Embed object*
onMouseOut, FIELDSET object*
onMouseOut, FileUpload object*
onMouseOut, FONT object*
onMouseOut, Form object*
onMouseOut, FormElement object*
onMouseOut, FRAMESET object*
onMouseOut, HEAD object*

Event handler (continued)

onMouseOut, H<n> object*
onMouseOut, HR object*
onMouseOut, HTML object*
onMouseOut, HyperLink object*
onMouseOut, I object*
onMouseOut, IFRAME object*
onMouseOut, Image object*
onMouseOut, IMG object*
onMouseOut, Input object*
onMouseOut, INS object*
onMouseOut, ISINDEX object*
onMouseOut, KBD object*
onMouseOut, Label object*
onMouseOut, Layer object*
onMouseOut, Legend object*
onMouseOut, LI object*
onMouseOut, LINK object*
onMouseOut, LISTING object*
onMouseOut, Location object*
onMouseOut, Map object*
onMouseOut, MARQUEE object*
onMouseOut, MENU object*
onMouseOut, META object*
onMouseOut, NOFRAMES object*
onMouseOut, NOSCRIPT object*
onMouseOut, OBJECT object*
onMouseOut, OL object*
onMouseOut, OptGroupElement object*
onMouseOut, Option object*
onMouseOut, P object*
onMouseOut, ParamElement object*
onMouseOut, Password object*
onMouseOut, PLAINTEXT object
onMouseOut, PRE object*
onMouseOut, Q object
onMouseOut, RadioButton object*
onMouseOut, ResetButton object*
onMouseOut, S object
onMouseOut, SAMP object
onMouseOut, SCRIPT object*
onMouseOut, Select object*
onMouseOut, SMALL object
onMouseOut, SPAN object*
onMouseOut, STRIKE object
onMouseOut, STRONG object*
onMouseOut, STYLE object (1)*
onMouseOut, style object (2)*
onMouseOut, SUB object
onMouseOut, SubmitButton object*
onMouseOut, SUP object
onMouseOut, TABLE object*
onMouseOut, TableColElement object*
onMouseOut, TBODY object*
onMouseOut, TD object*
onMouseOut, TEXTAREA object*
onMouseOut, TextCell object*
onMouseOut, TextRange object*
onMouseOut, TFOOT object*
onMouseOut, TH object*
onMouseOut, THEAD object*
onMouseOut, TITLE object*
onMouseOut, TR object*
onMouseOut, TT object*
onMouseOut, U object*
onMouseOut, UL object*
onMouseOut, Url object*
onMouseOut, VAR object
onMouseOver
onMouseOver, ! object*
onMouseOver, A object
onMouseOver, ABBR object
onMouseOver, ACRONYM object
onMouseOver, ADDRESS object
onMouseOver, Anchor object*
onMouseOver, Applet object*
onMouseOver, Area object*
onMouseOver, B object
onMouseOver, BASE object*
onMouseOver, BASEFONT object*
onMouseOver, BDO object*
onMouseOver, BGSOUND object*
onMouseOver, BIG object
onMouseOver, BLOCKQUOTE object*
onMouseOver, BODY object*
onMouseOver, BR object*
onMouseOver, Button object*
onMouseOver, BUTTON object*
onMouseOver, CAPTION object*
onMouseOver, CENTER object
onMouseOver, Checkbox object*
onMouseOver, CITE object
onMouseOver, CODE object
onMouseOver, COL object*
onMouseOver, COLGROUP object*
onMouseOver, DD object*
onMouseOver, DEL object*
onMouseOver, DFN object
onMouseOver, DIR object*
onMouseOver, DIV object*
onMouseOver, DL object*
onMouseOver, Document object*
onMouseOver, DT object*
onMouseOver, Element object*
onMouseOver, EM object*
onMouseOver, Embed object*
onMouseOver, FIELDSET object*
onMouseOver, FileUpload object*
onMouseOver, FONT object*
onMouseOver, Form object*
onMouseOver, FormElement object*
onMouseOver, FRAMESET object*
onMouseOver, HEAD object*
onMouseOver, H<n> object*
onMouseOver, HR object*
onMouseOver, HTML object*
onMouseOver, HyperLink object*
onMouseOver, I object*
onMouseOver, IFRAME object*
onMouseOver, Image object*
onMouseOver, IMG object*
onMouseOver, Input object*
onMouseOver, INS object*
onMouseOver, ISINDEX object*
onMouseOver, KBD object*
onMouseOver, Label object*
onMouseOver, Layer object*
onMouseOver, Legend object*
onMouseOver, LI object*
onMouseOver, LINK object*
onMouseOver, LISTING object*
onMouseOver, Location object*
onMouseOver, Map object*

onMouseOver, MARQUEE object*
onMouseOver, MENU object*
onMouseOver, META object*
onMouseOver, NOFRAMES object*
onMouseOver, NOSCRIPT object*
onMouseOver, OBJECT object*
onMouseOver, OL object*
onMouseOver, OptGroupElement object*
onMouseOver, Option object*
onMouseOver, P object*
onMouseOver, ParamElement object*
onMouseOver, Password object*
onMouseOver, PLAINTEXT object
onMouseOver, PRE object*
onMouseOver, Q object
onMouseOver, RadioButton object*
onMouseOver, ResetButton object*
onMouseOver, S object
onMouseOver, SAMP object
onMouseOver, SCRIPT object*
onMouseOver, Select object*
onMouseOver, SMALL object
onMouseOver, SPAN object*
onMouseOver, STRIKE object
onMouseOver, STRONG object*
onMouseOver, STYLE object (1)*
onMouseOver, style object (2)*
onMouseOver, SUB object
onMouseOver, SubmitButton object*
onMouseOver, SUP object
onMouseOver, TABLE object*
onMouseOver, TableColElement object*
onMouseOver, TBODY object*
onMouseOver, TD object*
onMouseOver, TEXTAREA object*
onMouseOver, TextCell object*
onMouseOver, TextRange object*
onMouseOver, TFOOT object*
onMouseOver, TH object*
onMouseOver, THEAD object*
onMouseOver, TITLE object*
onMouseOver, TR object*
onMouseOver, TT object*
onMouseOver, U object*
onMouseOver, UL object*
onMouseOver, Url object*
onMouseOver, VAR object
onMouseUp
onMouseUp, ! object*
onMouseUp, A object
onMouseUp, ABBR object
onMouseUp, ACRONYM object
onMouseUp, ADDRESS object
onMouseUp, Anchor object*
onMouseUp, Applet object*
onMouseUp, Area object*
onMouseUp, B object
onMouseUp, BASE object*
onMouseUp, BASEFONT object*
onMouseUp, BDO object*
onMouseUp, BGSOUND object*
onMouseUp, BIG object
onMouseUp, BLOCKQUOTE object*
onMouseUp, BODY object*
onMouseUp, BR object*
onMouseUp, Button object*
onMouseUp, BUTTON object*
onMouseUp, CAPTION object*
onMouseUp, CENTER object
onMouseUp, Checkbox object*
onMouseUp, CITE object
onMouseUp, CODE object
onMouseUp, COL object*
onMouseUp, COLGROUP object*
onMouseUp, DD object*
onMouseUp, DEL object*
onMouseUp, DFN object
onMouseUp, DIR object*
onMouseUp, DIV object*
onMouseUp, DL object*
onMouseUp, Document object*
onMouseUp, DT object*
onMouseUp, Element object*
onMouseUp, EM object*
onMouseUp, Embed object*
onMouseUp, FIELDSET object*
onMouseUp, FileUpload object*
onMouseUp, FONT object*
onMouseUp, Form object*
onMouseUp, FormElement object*
onMouseUp, FRAMESET object*
onMouseUp, HEAD object*
onMouseUp, H<n> object*
onMouseUp, HR object*
onMouseUp, HTML object*
onMouseUp, HyperLink object*
onMouseUp, I object*
onMouseUp, IFRAME object*
onMouseUp, Image object*
onMouseUp, IMG object*
onMouseUp, Input object*
onMouseUp, INS object*
onMouseUp, ISINDEX object*
onMouseUp, KBD object*
onMouseUp, Label object*
onMouseUp, Layer object*
onMouseUp, Legend object*
onMouseUp, LI object*
onMouseUp, LINK object*
onMouseUp, LISTING object*
onMouseUp, Location object*
onMouseUp, Map object*
onMouseUp, MARQUEE object*
onMouseUp, MENU object*
onMouseUp, META object*
onMouseUp, NOFRAMES object*
onMouseUp, NOSCRIPT object*
onMouseUp, OBJECT object*
onMouseUp, OL object*
onMouseUp, OptGroupElement object*
onMouseUp, Option object*
onMouseUp, P object*
onMouseUp, ParamElement object*
onMouseUp, Password object*
onMouseUp, PLAINTEXT object
onMouseUp, PRE object*
onMouseUp, Q object
onMouseUp, RadioButton object*
onMouseUp, ResetButton object*
onMouseUp, S object
onMouseUp, SAMP object
onMouseUp, SCRIPT object*
onMouseUp, Select object*
onMouseUp, SMALL object

Event handler (continued)

onMouseUp, SPAN object*
onMouseUp, STRIKE object
onMouseUp, STRONG object*
onMouseUp, STYLE object (1)*
onMouseUp, style object (2)*
onMouseUp, SUB object
onMouseUp, SubmitButton object*
onMouseUp, SUP object
onMouseUp, TABLE object*
onMouseUp, TableColElement object*
onMouseUp, TBODY object*
onMouseUp, TD object*
onMouseUp, TEXTAREA object*
onMouseUp, TextCell object*
onMouseUp, TextRange object*
onMouseUp, TFOOT object*
onMouseUp, TH object*
onMouseUp, THEAD object*
onMouseUp, TITLE object*
onMouseUp, TR object*
onMouseUp, TT object*
onMouseUp, U object*
onMouseUp, UL object*
onMouseUp, Url object*
onMouseUp, VAR object
onMove
onMove, Dialog object*
onMove, Frame object*
onMove, Global object*
onMove, Window object*
onPaste
onPaste, Document object*
onPropertyChange
onPropertyChange, Document object*
onReadyStateChange
onReadyStateChange, Applet object*
onReadyStateChange, Area object*
onReadyStateChange, Document object*
onReadyStateChange, LINK object*
onReadyStateChange, OBJECT object*
onReadyStateChange, SCRIPT object*
onReadyStateChange, STYLE object (1)*
onReadyStateChange, XML object*
onReset
onReset, Form object*
onResize
onResize, Applet object*
onResize, Area object*
onResize, BUTTON object*
onResize, Dialog object*
onResize, DIV object*
onResize, FIELDSET object*
onResize, FileUpload object*
onResize, Frame object*
onResize, FRAMESET object*
onResize, Global object*
onResize, IMG object*
onResize, MARQUEE object*
onResize, Password object*
onResize, Select object*
onResize, TABLE object*
onResize, TD object*
onResize, TH object*
onResize, Window object*
onRowEnter
onRowEnter, Applet object*
onRowEnter, Area object*
onRowEnter, BODY object*
onRowEnter, Button object*
onRowEnter, BUTTON object*
onRowEnter, Checkbox object*
onRowEnter, DIV object*
onRowEnter, Document object*
onRowEnter, FileUpload object*
onRowEnter, FormElement object*
onRowEnter, Hidden object*
onRowEnter, IMG object*
onRowEnter, Input object*
onRowEnter, MARQUEE object*
onRowEnter, OBJECT object*
onRowEnter, Password object*
onRowEnter, RadioButton object*
onRowEnter, ResetButton object*
onRowEnter, Select object*
onRowEnter, SubmitButton object*
onRowEnter, TABLE object*
onRowEnter, TD object*
onRowEnter, TEXTAREA object*
onRowEnter, TextCell object*
onRowEnter, TH object*
onRowEnter, XML object*
onRowExit
onRowExit, Applet object*
onRowExit, Area object*
onRowExit, BODY object*
onRowExit, Button object*
onRowExit, BUTTON object*
onRowExit, Checkbox object*
onRowExit, DIV object*
onRowExit, Document object*
onRowExit, FileUpload object*
onRowExit, FormElement object*
onRowExit, Hidden object*
onRowExit, IMG object*
onRowExit, Input object*
onRowExit, MARQUEE object*
onRowExit, OBJECT object*
onRowExit, Password object*
onRowExit, RadioButton object*
onRowExit, ResetButton object*
onRowExit, Select object*
onRowExit, SubmitButton object*
onRowExit, TABLE object*
onRowExit, TD object*
onRowExit, TEXTAREA object*
onRowExit, TextCell object*
onRowExit, TH object*
onRowExit, XML object*
onRowsDelete
onRowsDelete, XML object*
onRowsInserted
onRowsInserted, XML object*
onScroll
onScroll, BODY object*
onScroll, CAPTION object*
onScroll, Dialog object*
onScroll, DIV object*
onScroll, FIELDSET object*
onScroll, Frame object*
onScroll, IMG object*
onScroll, Legend object*
onScroll, MARQUEE object*
onScroll, TABLE object*

onScroll, TEXTAREA object*
onScroll, Window object*
onSelect
onSelect, CAPTION object*
onSelect, FIELDSET object*
onSelect, FileUpload object*
onSelect, FormElement object*
onSelect, Input object*
onSelect, Password object*
onSelect, TEXTAREA object*
onSelect, TextCell object*
onSelectStart
onSelectStart, A object
onSelectStart, ACRONYM object
onSelectStart, ADDRESS object
onSelectStart, B object
onSelectStart, BIG object
onSelectStart, BLOCKQUOTE object*
onSelectStart, BODY object*
onSelectStart, BUTTON object*
onSelectStart, CAPTION object*
onSelectStart, CENTER object
onSelectStart, CITE object
onSelectStart, CODE object
onSelectStart, DD object*
onSelectStart, DEL object*
onSelectStart, DFN object
onSelectStart, DIR object*
onSelectStart, DIV object*
onSelectStart, DL object*
onSelectStart, Document object*
onSelectStart, DT object*
onSelectStart, EM object*
onSelectStart, FIELDSET object*
onSelectStart, FileUpload object*
onSelectStart, FONT object*
onSelectStart, Form object*
onSelectStart, H<n> object*
onSelectStart, HR object*
onSelectStart, I object*
onSelectStart, IMG object*
onSelectStart, INS object*
onSelectStart, KBD object*
onSelectStart, Label object*
onSelectStart, Legend object*
onSelectStart, LI object*
onSelectStart, LISTING object*
onSelectStart, MARQUEE object*
onSelectStart, MENU object*
onSelectStart, OBJECT object*
onSelectStart, OL object*
onSelectStart, P object*
onSelectStart, Password object*
onSelectStart, PLAINTEXT object
onSelectStart, PRE object*
onSelectStart, Q object
onSelectStart, S object
onSelectStart, SAMP object
onSelectStart, Select object*
onSelectStart, SMALL object
onSelectStart, SPAN object*
onSelectStart, STRIKE object
onSelectStart, STRONG object*
onSelectStart, SUB object
onSelectStart, SUP object
onSelectStart, TABLE object*
onSelectStart, TBODY object*
onSelectStart, TD object*
onSelectStart, TEXTAREA object*
onSelectStart, TFOOT object*
onSelectStart, TH object*
onSelectStart, THEAD object*
onSelectStart, TR object*
onSelectStart, TT object*
onSelectStart, U object*
onSelectStart, UL object*
onSelectStart, VAR object
onStart
onStart, MARQUEE object*
onStop
onStop, Document object*
onSubmit
onSubmit, Form object*
onUnload
onUnload, BODY object*
onUnload, Dialog object*
onUnload, Frame object*
onUnload, FRAMESET object*
onUnload, Global object*
onUnload, Window object*

External code call

` (Backquote)
Backquote (`)*

File extension (see also Special file)

.cfg
.cgi
.htc*
.htm
.html
.jar*
.java*
.js*
.jsc
.jse
.jsh
.lck
.pac*
.shtm
.shtml
.stm
.web

Filter/blend

BlendTrans()
filter - BlendTrans()*

Filter/procedural

filter - AlphaImageLoader()*
filter - Gradient()*

Filter/reveal

filter - RevealTrans()*
RevealTrans()

Filter/transition

Barn()
Blinds()

Filter/transition (continued)

CheckerBoard()
Fade()
filter - Barn()*
filter - Blinds()*
filter - CheckerBoard()*
filter - Fade()*
filter - GradientWipe()*
filter - Inset()*
filter - Iris()*
filter - Pixelate()*
filter - RadialWipe()*
filter - RandomBars()*
filter - RandomDissolve()*
filter - Slide()*
filter - Spiral()*
filter - Stretch()*
filter - Strips()*
filter - Wheel()*
filter - Zigzag()*
GradientWipe()
Inset()
Iris()
Pixelate()
RadialWipe()
RandomBars()
RandomDissolve()
Slide()
Spiral()
Stretch()
Strips()
Wheel()
Zigzag()

Filter/visual

Alpha()
AlphaImageLoader()
BasicImage()
Blur()
Chroma()
Compositor()
DropShadow()
Emboss()
Engrave()
filter - Alpha()*
filter - BasicImage()*
filter - Blur()*
filter - Chroma()*
filter - Compositor()*
filter - DropShadow()*
filter - Emboss()*
filter - Engrave()*
filter - FlipH()*
filter - FlipV()*
filter - Glow()*
filter - Grayscale()*
filter - Invert()*
filter - Light()*
filter - Mask()*
filter - MaskFilter()*
filter - Matrix()*
filter - MotionBlur()*
filter - Pixelate()*
filter - Shadow()*
filter - Wave()*
filter - XRay()*
FlipH()
FlipV()
Glow()
Gradient()
Grayscale()
Invert()
Light()
Mask()
MaskFilter()
Matrix()
MotionBlur()
Pixelate()
Shadow()
Wave()
XRay()

Function

abs()*, Math.abs()*
acos()*, Math.acos()*
Array()*
asin()*, Math.asin()*
atan()*, Math.atan()*
atan2()*, Math.atan2()*
atob()*, Window.atob()*
Boolean()*
btoa()*, Window.btoa()*
captureEvents()*
captureEvents()*,
  Document.captureEvents()*
captureEvents()*, Layer.captureEvents()*
captureEvents()*,
  Window.captureEvents()*
catch(...)*
ceil()*, Math.ceil()*
cos()*, Math.cos()*
Date()*
decodeURI()*
decodeURIComponent()*
encodeURI()*
encodeURIComponent()*
Error()*
exp()*, Math.exp()*
floor()*, Math.floor()*
Function()*
getClass()
GetObject()*
handleEvent()*
handleEvent()*, Document.handleEvent()*
handleEvent()*, Layer.handleEvent()*
handleEvent()*, Window.handleEvent()*
Image()
log()*, Math.log()*
max()*, Math.max()*
min()*, Math.min()*
Number()*
Object()*
pow()*, Math.pow()*
random(), Crypto.random()
random()*, Math.random()*
RegExp()
releaseEvents()
releaseEvents()*,
  Document.releaseEvents()*
releaseEvents()*, Layer.releaseEvents()*
releaseEvents()*, Window.releaseEvents()*
rgb()*
round()*, Math.round()*

routeEvent()
routeEvent(), Document.routeEvent()
routeEvent()*, Layer.routeEvent()*
routeEvent()*, Window.routeEvent()*
ScriptEngine()*
signText(), Crypto.signText()
sin()*, Math.sin()*
sqrt()*, Math.sqrt()*
String()*
tan()*, Math.tan()*

Function/global

escape()*
eval()*
isFinite()*
isNaN()*
parseFloat()*
parseInt()*
ScriptEngineBuildVersion()*
ScriptEngineMajorVersion()
ScriptEngineMinorVersion()
taint()*
toString()*
unescape()*
untaint()*
unwatch()*
watch()*

Function/internal

Call*
CanPut()*
DefaultValue()*
Delete()*
Get()*
GetBase()*
GetPropertyName()*
GetValue()*
HasInstance()*
HasProperty()*
Put()*
PutValue()*

Function/proxy.pac

FindProxyForURL()*
isInNet()*
isPlainHostName()*

HTML Tag

<EMBED>*
<META>*
<NOSCRIPT>*
<SCRIPT>*
<STYLE>*
<TITLE>*
Conditional comment*
HTML Comment tag (<!-- ... -->)

HTML Tag Attribute

<MAP TARGET="...">*
<SCRIPT ARCHIVE="...">*
<SCRIPT EVENT="...">*
<SCRIPT FOR="...">
<SCRIPT ID="...">*
<SCRIPT LANGUAGE="...">*
<SCRIPT SRC="...">*
<SCRIPT TYPE="...">*
<STYLE TYPE="...">*
CLASS="..."*
HTTP-EQUIV="..."
ID="..."*
LANG="..."*
MAYSCRIPT*
NAME="..."

Interface

Error handler*
Watchpoint handler*

Iterator

do ... while(...)*
for(...) ...*
for(... in ...) ...*
while(...) ...*

Java class

java.awt.Button*
java.awt.image*
java.lang.Boolean*
java.lang.Character*
java.lang.Class*
java.lang.Double*
java.lang.Float*
java.lang.Integer*
java.lang.Long*
java.lang.Object*
java.lang.String*
java.util.Date*
JSObject object*
netscape.javascript.JSObject*
netscape.plugin.Plugin
netscape.security.PrivilegeManager*
PrivilegeManager object*

Java method

call()*, JSObject.call()*
eval()*, JSObject.eval()*
getMember()*, JSObject.getMember()*
getSlot()*, JSObject.getSlot()*
removeMember()*,
JSObject.removeMember()*
setMember()*, JSObject.setMember()*
setSlot()*, JSObject.setSlot()*
toString()*, JSObject.toString()*

Java method/static

JSObject.getWindow()*

Java package

java.awt*
java.lang*
java.util*
netscape
netscape.applet*
netscape.cfg
netscape.javascript
netscape.lck*
netscape.plugin

Java package (continued)

netscape.security
Packages.java*
Packages.netscape*
Packages.netscape.javascript
Packages.netscape.plugin
Packages.sun
sun

Keyword

else ...*
in ...
this*

Label

case ... :
default:

Method

action(), java.awt.Button*
add(), Area object*
add(), Area.add()
Add(), Dictionary object*
Add(), Dictionary.Add()
Add(), Folders object*
Add(), Folders.Add()
add(), OptionsArray object*
add(), OptionsArray.add()
add(), Select object*
add(), Select.add()
addAmbient(), filter - Light()*
addBehavior(), Element object*
addBehavior(), Element.addBehavior()
AddChannel(), external object*
AddChannel(), external.AddChannel()
addClient(), Global object*
addClient(), response object*
addClient(), response.addClient()
addCone(), filter - Light()*
AddDesktopComponent(), external object*
AddDesktopComponent(),
external.AddDesktopComponent()
addEventListener(), EventTarget object*
addEventListener(),
EventTarget.addEventListener()
AddFavorite(), external object*
AddFavorite(), external.AddFavorite()
addImport(), StyleSheet object*
addImport(), StyleSheet.addImport()
addNotify(), java.awt.Button*
addPoint(), filter - Light()*
addReadRequest(), userProfile object*
addReadRequest(),
userProfile.addReadRequest()
addResponseHeader(), Global object*
addResponseHeader(), response object*
addResponseHeader(),
response.addResponseHeader()
addRule(), StyleSheet object*
addRule(), StyleSheet.addRule()
after(), java.util.Date*
alert()
alert(), Global object*
alert(), Window object*
alert(), Window.alert()
anchor(), String object*
anchor(), String.anchor()
appendChild(), Node object*
appendChild(), Node.appendChild()
appendData(), CharacterData object*
appendData(), CharacterData.appendData()
apply(), filter - Barn()*
apply(), filter - Blinds()*
apply(), filter - Compositor()*
apply(), filter - Fade()*
apply(), filter - GradientWipe()*
apply(), filter - Inset()*
apply(), filter - Iris()*
apply(), filter - Pixelate()*
apply(), filter - Pixelate()*
apply(), filter - RadialWipe()*
apply(), filter - RandomBars()*
apply(), filter - RandomDissolve()*
apply(), filter - Slide()*
apply(), filter - Spiral()*
apply(), filter - Stretch()*
apply(), filter - Strips()*
apply(), filter - Wheel()*
apply(), filter - Zigzag()*
apply(), Function object*
apply(), Function.apply()
applyElement(), Element object*
applyElement(), Element.applyElement()
Array()*
Array(), Global object*
assign(), Location object*
assign(), Location.assign()
assign(), Object object*
assign(), Object.assign()
atEnd(), Enumerator object*
atEnd(), Enumerator.atEnd()
atob()
atob()*, Window.atob()*
attachEvent()
attachEvent(), Document object*
attachEvent(), Document.attachEvent()
attachEvent(), Global object*
attachEvent(), Window object*
attachEvent(), Window.attachEvent()
AutoCompleteSaveForm(), external object*
AutoCompleteSaveForm(),
external.AutoCompleteSaveForm()
AutoScan(), external object*
AutoScan(), external.AutoScan()
back()
back(), Global object*
back(), History object*
back(), History.back()
back(), Window object*
back(), Window.back()
before(), java.util.Date*
beginTransaction(), Connection object*
beginTransaction(),
Connection.beginTransaction()
beginTransaction(), database object*
beginTransaction(),
database.beginTransaction()
big(), String object*
big(), String.big()
blink(), String object*
blink(), String.blink()
blob(), Global object*

Method (continued)

compareEndPoints(),
TextRange.compareEndPoints()
compareTo(), java.lang.String*
compile(), RegExp object*
compile(), RegExp.compile()
componentFromPoint(), Element object*
componentFromPoint(),
Element.componentFromPoint()
concat(), Array object*
concat(), Array.concat()
concat(), java.lang.String*
concat(), String object*
concat(), String.concat()
confirm()
confirm(), Global object*
confirm(), Window object*
confirm(), Window.confirm()
connect(), database object*
connect(), database.connect()
connect(), DbPool object*
connect(), DbPool.connect()
connected(), Connection object*
connected(), Connection.connected()
connected(), database object*
connected(), database.connected()
connected(), DbPool object*
connected(), DbPool.connected()
connection(), DbPool object*
connection(), DbPool.connection()
contains(), Element object*
contains(), Element.contains()
contains(), Frame object*
contextual()
contextual(), Document object*
contextual(), Document.contextual()
Copy(), File object*
Copy(), File.Copy()
Copy(), Folder object*
Copy(), Folder.Copy()
CopyFile(), FileSystem object*
CopyFile(), FileSystem.CopyFile()
CopyFolder(), FileSystem object*
CopyFolder(), FileSystem.CopyFolder()
createAttribute(), Document object*
createAttribute(),
Document.createAttribute()
createCaption(), TABLE object*
createCaption(), TABLE.createCaption()
createCDATASection(), Document object*
createCDATASection(),
Document.createCDATASection()
createComment(), Document object*
createComment(),
Document.createComment()
createControlRange(), BODY object*
createControlRange(),
BODY.createControlRange()
createDocumentFragment(), Document
object*
createDocumentFragment(),
Document.createDocumentFragment()
createElement(), Document object*
createElement(),
Document.createElement()
createEntityReference(), Document object*
createEntityReference(),
Document.createEntityReference()
createEvent(), DocumentEvent*
createEvent(),
DocumentEvent.createEvent()
CreateFolder(), FileSystem object*
CreateFolder(), FileSystem.CreateFolder()
createImage(), java.awt.Button*
CreateObject(), WScript object*
CreateObject(), WScript.CreateObject()
createProcessingInstruction(), Document
object*
createProcessingInstruction(),
Document.createProcessingInstruction()
createRange(), Selection object*
createRange(), selection.createRange()
createStyleSheet(), Document object*
createStyleSheet(),
Document.createStyleSheet()
CreateTextFile(), FileSystem object*
CreateTextFile(),
FileSystem.CreateTextFile()
createTextNode(), Document object*
createTextNode(),
Document.createTextNode()
createTextRange(), BODY object*
createTextRange(),
BODY.createTextRange()
createTextRange(), BUTTON object*
createTextRange(), Input object*
createTextRange(), Input.createTextRange()
createTFoot(), TABLE object*
createTFoot(), TABLE.createTFoot()
createTHead(), TABLE object*
createTHead(), TABLE.createTHead()
cursor(), Connection object*
cursor(), Connection.cursor()
cursor(), database object*
cursor(), database.cursor()
Date()*
Date(), Global object*
debug(), Global object*
debug(), response object*
debug(), response.debug()
Delete(), File object*
Delete(), File.Delete()
Delete(), Folder object*
Delete(), Folder.Delete()
deleteCaption(), TABLE object*
deleteCaption(), TABLE.deleteCaption()
deleteCell(), TR object*
deleteCell(), TR.deleteCell()
deleteData(), CharacterData object*
deleteData(), CharacterData.deleteData()
DeleteFile(), FileSystem object*
DeleteFile(), FileSystem.DeleteFile()
DeleteFolder(), FileSystem object*
DeleteFolder(), FileSystem.DeleteFolder()
deleteResponseHeader(), Global object*
deleteResponseHeader(), response object*
deleteResponseHeader(),
response.deleteResponseHeader()
deleteRow(), Cursor object*
deleteRow(), Cursor.deleteRow()
deleteRow(), TABLE object*
deleteRow(), TABLE.deleteRow()

Method (continued)

find(), Window.find()
findText(), TextRange object*
findText(), TextRange.findText()
fixed(), String object*
fixed(), String.fixed()
floatValue(), java.lang.Double*
floatValue(), java.lang.Float*
floatValue(), java.lang.Integer*
floatValue(), java.lang.Long*
flush, Global object*
flush(), File object*
flush(), File object*
flush(), File.flush()
flush(), response object*
flush(), response.flush()
focus()
focus(), Anchor object*
focus(), Anchor.focus()
focus(), Element object*
focus(), File object*
focus(), Global object*
focus(), Input object*
focus(), Input.focus()
focus(), Window object*
focus(), Window.focus()
FolderExists(), FileSystem object*
FolderExists(), FileSystem.FolderExists()
fontcolor(), String object*
fontcolor(), String.fontcolor()
fontsize(), String object*
fontsize(), String.fontsize()
forward()
forward(), Global object*
forward(), History object*
forward(), History.forward()
forward(), Window object*
forward(), Window.forward()
fromCharCode(), String object*
fromCharCode()*, String.fromCharCode()*
Function()*
Function(), Global object*
GetAbsolutePathName(), FileSystem object*
GetAbsolutePathName(), FileSystem.GetAbsolutePathName()
getAdjacentText(), Element object*
getAdjacentText(), Element.getAdjacentText()
getAppletContext(), JavaObject object*
getAppletInfo(), JavaObject object*
getAttribute(), BASEFONT object*
getAttribute(), currentStyle object*
getAttribute(), Element object*
getAttribute(), Element.getAttribute()
getAttribute(), Frame object*
getAttribute(), runtimeStyle object*
getAttribute(), style object (2)*
getAttribute(), style.getAttribute()
getAttribute(), userProfile object*
getAttribute(), userProfile.getAttribute()
getAttributeNode(), Element object*
getAttributeNode(), Element.getAttributeNode()
getBackground(), java.awt.Button*
getBackground(), JavaObject object*
GetBaseName(), FileSystem object*
GetBaseName(), FileSystem.GetBaseName()
getBookmark(), TextRange object*
getBookmark(), TextRange.getBookmark()
getBoundingClientRect(), TextRange object*
getBoundingClientRect(), TextRange.getBoundingClientRect()
getBytes(), java.lang.String*
getChars(), java.lang.String*
getClass()
getClass(), java.awt.Button*
getClass(), java.lang.Boolean*
getClass(), java.lang.Character*
getClass(), java.lang.Double*
getClass(), java.lang.Float*
getClass(), java.lang.Integer*
getClass(), java.lang.Long*
getClass(), java.lang.Object*
getClass(), java.lang.String*
getClass(), java.util.Date*
getClass(), JavaObject object*
getClass(), JavaObject.getClass()*
getClass(), netscape.plugin.Plugin
getClientRects(), TextRange object*
getClientRects(), TextRange.getClientRects()
getCodeBase(), JavaObject object*
getColorModel(), java.awt.Button*
getData(), dataTransfer object*
getData(), dataTransfer.getData()
getDate(), Date object*
getDate(), Date.getDate()
getDate(), java.util.Date*
getDay(), Date object*
getDay(), Date.getDay()
getDay(), java.util.Date*
getDocumentBase(), JavaObject object*
GetDrive(), FileSystem object*
GetDrive(), FileSystem.GetDrive()
GetDriveName(), FileSystem object*
GetDriveName(), FileSystem.GetDriveName()
getElementById(), Document object*
getElementById(), Document.getElementById()
getElementsByName(), Document object*
getElementsByName(), Document.getElementsByName()
getElementsByTagName(), Document object*
getElementsByTagName(), Document.getElementsByTagName()
getElementsByTagName(), Element object*
getElementsByTagName(), Element.getElementsByTagName()
getExpression(), currentStyle object*
getExpression(), Element object*
getExpression(), Element.getExpression()
getExpression(), runtimeStyle object*
getExpression(), style object (2)*
getExpression(), style.getExpression()
GetExtensionName(), FileSystem object*
GetExtensionName(), FileSystem.GetExtensionName()
GetFile(), FileSystem object*

GetFile(), FileSystem.GetFile()
GetFileName(), FileSystem object*
GetFileName(), FileSystem.GetFileName()
GetFolder(), FileSystem object*
GetFolder(), FileSystem.GetFolder()
getFont(), java.awt.Button*
getFontMetrics(), java.awt.Button*
getForeground(), java.awt.Button*
getFullYear(), Date object*
getFullYear(), Date.getFullYear()
getGraphics(), java.awt.Button*
getHours(), Date object*
getHours(), Date.getHours()
getHours(), java.util.Date*
getItem(), VBArray object*
getItem(), VBArray.getItem()
getLabel(), java.awt.Button*
getLength(), File object*
getLength(), File.getLength()
getLocale(), java.awt.Button*
getLocale(), JavaObject object*
getMember(), JSObject object*
getMember(), JSObject.getMember()*
getMilliseconds(), Date object*
getMilliseconds(), Date.getMilliseconds()
getMinutes(), Date object*
getMinutes(), Date.getMinutes()
getMinutes(), java.util.Date*
getMonth(), Date object*
getMonth(), Date.getMonth()
getMonth(), java.util.Date*
getNamedItem(), NamedNodeMap object*
getNamedItem(),
  NamedNodeMap.getNamedItem()
GetObject()*
GetObject(), WScript object*
GetObject(), WScript.GetObject()
getOptionValue(), Global object*
getOptionValue(), response object*
getOptionValue(),
  response.getOptionValue()
getOptionValueCount(), Global object*
getOptionValueCount(), response object*
getOptionValueCount(),
  response.getOptionValueCount()
getParameter(), JavaObject object*
getParameterInfo(), JavaObject object*
getParent(), java.awt.Button*
GetParentFolderName(), FileSystem
  object*
GetParentFolderName(),
  FileSystem.GetParentFolderName()
getPeer(), java.awt.Button*
getPeer(), netscape.plugin.Plugin
getPosition(), File object*
getPosition(), File.getPosition()
getPrivilegeTableFromStack(),
  netscape.security.PrivilegeManager*
getSeconds(), Date object*
getSeconds(), Date.getSeconds()
getSeconds(), java.util.Date*
getSelection(), Document object*
getSelection(), Document.getSelection()
getSlot(), JSObject object*
getSlot(), JSObject.getSlot()*
GetSpecialFolder(), FileSystem object*
GetSpecialFolder(),
  FileSystem.GetSpecialFolder()
GetTempName(), FileSystem object*
GetTempName(),
  FileSystem.GetTempName()
getTime(), Date object*
getTime(), Date.getTime()
getTime(), java.util.Date*
getTimezoneOffset(), Date object*
getTimezoneOffset(),
  Date.getTimezoneOffset()
getTimezoneOffset(), java.util.Date*
getToolkit(), java.awt.Button*
getToolkit(), JavaObject object*
getUTCDate(), Date object*
getUTCDate(), Date.getUTCDate()
getUTCDay(), Date object*
getUTCDay(), Date.getUTCDay()
getUTCFullYear(), Date object*
getUTCFullYear(), Date.getUTCFullYear()
getUTCHours(), Date object*
getUTCHours(), Date.getUTCHours()
getUTCMilliseconds(), Date object*
getUTCMilliseconds(),
  Date.getUTCMilliseconds()
getUTCMinutes(), Date object*
getUTCMinutes(), Date.getUTCMinutes()
getUTCMonth(), Date object*
getUTCMonth(), Date.getUTCMonth()
getUTCSeconds(), Date object*
getUTCSeconds(), Date.getUTCSeconds()
getVarDate(), Date object*
getVarDate(), Date.getVarDate()
getWindow(), netscape.plugin.Plugin
getWindow(), JSObject.getWindow()*
getYear(), Date object*
getYear(), Date.getYear()
getYear(), java.util.Date*
go(), History object*
go(), History.go()
gotFocus(), java.awt.Button*
handleEvent()*
handleEvent(), Button object*
handleEvent(), Button.handleEvent()
handleEvent(), Checkbox object*
handleEvent(), Checkbox.handleEvent()
handleEvent(), Document object*
handleEvent(), Document.handleEvent()*
handleEvent(), FileUpload object*
handleEvent(), FileUpload.handleEvent()
handleEvent(), Form object*
handleEvent(), Form.handleEvent()
handleEvent(), Input object*
handleEvent(), Input.handleEvent()
handleEvent(), Layer.handleEvent()*
handleEvent(), Password object*
handleEvent(), Password.handleEvent()
handleEvent(), RadioButton object*
handleEvent(), RadioButton.handleEvent()
handleEvent(), ResetButton object*
handleEvent(), ResetButton.handleEvent()
handleEvent(), SubmitButton object*
handleEvent(), SubmitButton.handleEvent()
handleEvent(), TEXTAREA object*
handleEvent(), TEXTAREA.handleEvent()
handleEvent(), TextCell object*

Method (continued)

handleEvent(), TextCell.handleEvent()
handleEvent(), Window.handleEvent()*
hasChildNodes(), Node object*
hasChildNodes(), Node.hasChildNodes()
hasFeature(), Implementation object*
hasFeature(), Implementation.hasFeature()
hashCode(), java.awt.Button*
hashCode(), java.lang.Boolean*
hashCode(), java.lang.Character*
hashCode(), java.lang.Double*
hashCode(), java.lang.Float*
hashCode(), java.lang.Integer*
hashCode(), java.lang.Long*
hashCode(), java.lang.Object*
hashCode(), java.lang.String*
hashCode(), java.util.Date*
hashCode(), netscape.plugin.Plugin
hasOwnProperty(), Object object*
hasOwnProperty(), Object.hasOwnProperty()
hide(), java.awt.Button*
hide(), JavaObject object*
home()
home(), Global object*
home(), Window object*
home(), Window.home()
imageUpdate(), java.awt.Button*
ImportExportFavorites(), external object*
ImportExportFavorites(),
external.ImportExportFavorites()
indexOf(), java.lang.String*
indexOf(), String object*
indexOf(), String.indexOf()
init(), JavaObject object*
init(), netscape.plugin.Plugin
initEvent(), Event object*
initEvent(), Event.initEvent()
initEvent(), MouseEvent object*
initEvent(), MutationEvent object*
initMouseEvent(), MouseEvent object*
initMouseEvent(),
MouseEvent.initMouseEvent()
initMutationEvent(), MutationEvent object*
initMutationEvent(),
MutationEvent.initMutationEvent()
initUIEvent(), MouseEvent object*
initUIEvent(), UIEvent object*
initUIEvent(), UIEvent.initUIEvent()
inRange(), TextRange object*
inRange(), TextRange.inRange()
insertAdjacentHTML(), Element object*
insertAdjacentHTML(),
Element.insertAdjacentHTML()
insertAdjacentText(), Element object*
insertAdjacentText(),
Element.insertAdjacentText()
insertBefore(), Node object*
insertBefore(), Node.insertBefore()
insertCell(), TR object*
insertCell(), TR.insertCell()
insertData(), CharacterData object*
insertData(), CharacterData.insertData()
insertRow(), Cursor object*
insertRow(), Cursor.insertRow()
insertRow(), TABLE object*
insertRow(), TABLE.insertRow()
insertRow(), TFOOT object*
insertRow(), TFOOT.insertRow()
insertRow(), THEAD object*
insertRow(), THEAD.insertRow()
inside(), java.awt.Button*
intern(), java.lang.String*
intValue(), java.lang.Double*
intValue(), java.lang.Float*
intValue(), java.lang.Integer*
intValue(), java.lang.Long*
invalidate(), java.awt.Button*
isActive(), JavaObject object*
isActive(), netscape.plugin.Plugin
isActive(), Plugin object*
isActive(), Plugin.isActive()
isEnabled(), java.awt.Button*
isEnabled(), JavaObject object*
isEqual(), TextRange object*
isEqual(), TextRange.isEqual()
isFinite(), Global object*
isInfinite(), java.lang.Double*
isInfinite(), java.lang.Float*
isNaN(), Global object*
isNaN(), java.lang.Double*
isNaN(), java.lang.Float*
isPrototypeOf(), Object object*
isPrototypeOf(), Object.isPrototypeOf()
isShowing(), java.awt.Button*
isShowing(), JavaObject object*
IsSubscribed(), external object*
IsSubscribed(), external.IsSubscribed()
isValid(), java.awt.Button*
isValid(), JavaObject object*
isValid(), Lock object*
isValid(), Lock.isValid()
isVisible(), java.awt.Button*
isVisible(), JavaObject object*
italics(), String object*
italics(), String.italics()
Item(), Collection object*
Item(), Collection.Item()
Item(), Dictionary object*
Item(), Dictionary.Item()
item(), Enumerator object*
item(), Enumerator.item()
Item(), Files object*
Item(), Files.Item()
item(), Filters object
item(), Filters.item()
Item(), Folders object*
Item(), Folders.Item()
item(), FormArray object*
item(), FormArray.item()
item(), FrameArray object
item(), FrameArray.item()
item(), Frames object*
item(), ImageArray object*
item(), ImageArray.item()
item(), NamedNodeMap object*
item(), NamedNodeMap.item()
item(), NodeList object*
item(), NodeList.item()
item(), OptionsArray object*
item(), OptionsArray.item()
item(), PluginArray object*
item(), PluginArray.item()
item(), rows object
item(), ScriptArray object*

Method (continued)

moveToAbsolute(), Layer.moveToAbsolute()
moveToBookmark(), TextRange object*
moveToBookmark(),
TextRange.moveToBookmark()
moveToElementText(), TextRange object*
moveToElementText(),
TextRange.moveToElementText()
moveToPoint(), TextRange object*
moveToPoint(), TextRange.moveToPoint()
namedItem(), Collection object*
namedItem(), Collection.namedItem()
navigate()
navigate(), Global object*
navigate(), Window object*
navigate(), Window.navigate()
NavigateAndFind(), external object*
NavigateAndFind(),
external.NavigateAndFind()
next(), Cursor object*
next(), Cursor.next()
next(), ResultSet object*
next(), ResultSet.next()
nextFocus(), java.awt.Button*
nextPage(), TABLE object*
nextPage(), TABLE.nextPage()
normalize(), Element object*
normalize(), Element.normalize()
notify(), java.awt.Button*
notify(), java.lang.Boolean*
notify(), java.lang.Character*
notify(), java.lang.Double*
notify(), java.lang.Float*
notify(), java.lang.Integer*
notify(), java.lang.Long*
notify(), java.lang.Object*
notify(), java.lang.String*
notify(), java.util.Date*
notify(), netscape.plugin.Plugin
notifyAll(), java.awt.Button*
notifyAll(), java.lang.Boolean*
notifyAll(), java.lang.Character*
notifyAll(), java.lang.Double*
notifyAll(), java.lang.Float*
notifyAll(), java.lang.Integer*
notifyAll(), java.lang.Long*
notifyAll(), java.lang.Object*
notifyAll(), java.lang.String*
notifyAll(), java.util.Date*
notifyAll(), netscape.plugin.Plugin
Number()*
Number(), Global object*
Object()*
Object(), Global object*
offset(), Layer object*
offset(), Layer.offset()
open()
open(), Document object*
open(), Document.open()
open(), File object*
open(), File object*
open(), File.open()
open(), Global object*
open(), Window object*
open(), Window.open()
OpenAsTextStream(), File object*
OpenAsTextStream(),
File.OpenAsTextStream()
OpenTextFile(), FileSystem object*
OpenTextFile(), FileSystem.OpenTextFile()
outParamCount(), Stproc object*
outParamCount(), Stproc.outParamCount()
outParameters(), Stproc object*
outParameters(), Stproc.outParameters()
paddings(), JSSTag object*
paddings(), JSSTag.paddings()
paint(), java.awt.Button*
paintAll(), java.awt.Button*
parentElement(), TextRange object*
parentElement(),
TextRange.parentElement()
parse(), Date object*
parse(), Date.parse()*
parseFloat(), Global object*
parseInt(), Global object*
pasteHTML(), TextRange object*
pasteHTML(), TextRange.pasteHTML()
play(), filter - Barn()*
play(), filter - Blinds()*
play(), filter - Compositor()*
play(), filter - Fade()*
play(), filter - GradientWipe()*
play(), filter - Inset()*
play(), filter - Iris()*
play(), filter - Pixelate()*
play(), filter - Pixelate()*
play(), filter - RadialWipe()*
play(), filter - RandomBars()*
play(), filter - RandomDissolve()*
play(), filter - Slide()*
play(), filter - Spiral()*
play(), filter - Stretch()*
play(), filter - Strips()*
play(), filter - Wheel()*
play(), filter - Zigzag()*
plugins, Navigator object*
pop(), Array object*
pop(), Array.pop()
postEvent(), java.awt.Button*
preference(), Navigator object*
preference(), Navigator.preference()
preferredSize(), java.awt.Button*
prepareImage(), java.awt.Button*
preventDefault(), Event object*
preventDefault(), Event.preventDefault()
preventDefault(), MouseEvent object*
preventDefault(), MutationEvent object*
previousPage(), TABLE object*
previousPage(), TABLE.previousPage()
print()
print(), Global object*
print(), java.awt.Button*
print(), Window object*
print(), Window.print()
printAll(), java.awt.Button*
prompt()
prompt(), Global object*
prompt(), Window object*
prompt(), Window.prompt()
propertyIsEnumerable(), Object object*
propertyIsEnumerable(),
Object.propertyIsEnumerable()

Method (continued)

removeNotify(), java.awt.Button*
removeRule(), StyleSheet object*
removeRule(), StyleSheet.removeRule()
repaint(), java.awt.Button*
replace(), java.lang.String*
replace(), Location object*
replace(), Location.replace()
replace(), String object*
replace(), String.replace()
replaceAdjacentText(), Element object*
replaceAdjacentText(),
Element.replaceAdjacentText()
replaceChild(), Node object*
replaceChild(), Node.replaceChild()
replaceData(), CharacterData object*
replaceData(), CharacterData.replaceData()
requestFocus(), java.awt.Button*
reset(), Form object*
reset(), Form.reset()
reshape(), java.awt.Button*
resize(), java.awt.Button*
resizeBy()
resizeBy(), Global object*
resizeBy(), Layer object*
resizeBy(), Layer.resizeBy()
resizeBy(), Window object*
resizeBy(), Window.resizeBy()
resizeTo()
resizeTo(), Global object*
resizeTo(), Layer object*
resizeTo(), Layer.resizeTo()
resizeTo(), Window object*
resizeTo(), Window.resizeTo()
resultSet(), Stproc object*
resultSet(), Stproc.resultSet()
returnValue(), Stproc object*
returnValue(), Stproc.returnValue()
reverse(), Array object*
reverse(), Array.reverse()
rgb(), JSSTag object*
rgb(), JSSTag.rgb()
rollbackTransaction(), Connection object*
rollbackTransaction(),
Connection.rollbackTransaction()
rollbackTransaction(), database object*
rollbackTransaction(),
database.rollbackTransaction()
routeEvent(), Document object*
savePreferences(), Navigator object*
savePreferences(),
Navigator.savePreferences()
scroll()
scroll(), Global object*
scroll(), Window object*
scroll(), Window.scroll()
scrollBy()
scrollBy(), Global object*
scrollBy(), Window object*
scrollBy(), Window.scrollBy()
scrollIntoView(), Element object*
scrollIntoView(), Element.scrollIntoView()
scrollTo()
scrollTo(), Global object*
scrollTo(), Window object*
scrollTo(), Window.scrollTo()
search(), String object*
search(), String.search()
select(), File object*
select(), FileUpload object*
select(), FileUpload.select()
select(), Image object*
select(), IMG object*
select(), Input object*
select(), Input.select()
select(), OptionsArray object*
select(), OptionsArray.select()
select(), Password object*
select(), Password.select()
select(), TEXTAREA object*
select(), TEXTAREA.select()
select(), TextCell object*
select(), TextCell.select()
select(), TextRange object*
select(), TextRange.select()
send(), SendMail object*
send(), SendMail.send()
setAttribute(), COMMENT object*
setAttribute(), currentStyle object*
setAttribute(), Element object*
setAttribute(), Element.setAttribute()
setAttribute(), Frame object*
setAttribute(), runtimeStyle object*
setAttribute(), style object (2)*
setAttribute(), style.setAttribute()
setAttributeNode(), Element object*
setAttributeNode(),
Element.setAttributeNode()
setBackground(), java.awt.Button*
setCapture(), Element object*
setCapture(), Element.setCapture()
setData(), clipboardData object*
setData(), dataTransfer object*
setData(), dataTransfer.setData()
setDate(), Date object*
setDate(), java.util.Date*
setDate(), Date.setDate()
setEndPoint(), TextRange object*
setEndPoint(), TextRange.setEndPoint()
setExpression(), currentStyle object*
setExpression(), Element object*
setExpression(), Element.setExpression()
setExpression(), runtimeStyle object*
setExpression(), style object (2)*
setExpression(), style.setExpression()
setFont(), java.awt.Button*
setForeground(), java.awt.Button*
setFullYear(), Date object*
setFullYear(), Date.setFullYear()
setHotkeys()
setHotkeys(), Global object*
setHotkeys(), Window object*
setHotkeys(), Window.setHotkeys()
setHours(), Date object*
setHours(), Date.setHours()
setHours(), java.util.Date*
setInterval()
setInterval(), Global object*
setInterval(), Window object*
setInterval(), Window.setInterval()
setLabel(), java.awt.Button*
setMember(), JSObject object*
setMember(), JSObject.setMember()*
setMilliseconds(), Date object*

Method (continued)

toString(), JavaArray.toString()
toString(), JavaObject object*
toString(), JSObject object*
toString(), JSObject.toString()*
toString(), netscape.plugin.Plugin
toString(), Number object*
toString(), Number.toString()
toString(), Object object*
toString(), Object.toString()
toString(), RegExp object*
toString(), RegExp.toString()
toString(), String object*
toString(), String.toString()
toString(), URIError object*
toTimeString(), Date object*
toTimeString(), Date.toTimeString()
toUpperCase(), java.lang.String*
toUpperCase(), String object*
toUpperCase(), String.toUpperCase()
toUTCString(), Date object*
toUTCString(), Date.toUTCString()
trace(), response object*
trace(), response.trace()
trim(), java.lang.String*
typeof(), Global object*
ubound(), VBArray object*
ubound(), VBArray.ubound()
unescape(), Global object*
unlock(), Lock object*
unlock(), Lock.unlock()
unlock(), project object*
unlock(), project.unlock()
unlock(), server object*
unlock(), server.unlock()
unshift(), Array object*
unshift(), Array.unshift()
unwatch(), Object object*
unwatch(), Object.unwatch()
update(), java.awt.Button*
updateRow(), Cursor object*
updateRow(), Cursor.updateRow()
UTC(), Date.UTC()*
validate(), java.awt.Button*
valueOf()
valueOf(), Array object*
valueOf(), Array.valueOf()
valueOf(), Boolean object*
valueOf(), Boolean.valueOf()
valueOf(), Date object*
valueOf(), Date.valueOf()
valueOf(), Function object*
valueOf(), Function.valueOf()
valueOf(), Number object*
valueOf(), Number.valueOf()
valueOf(), Object object*
valueOf(), Object.valueOf()
valueOf(), String object*
valueOf(), String.valueOf()
wait(), java.awt.Button*
wait(), java.lang.Boolean*
wait(), java.lang.Character*
wait(), java.lang.Double*
wait(), java.lang.Float*
wait(), java.lang.Integer*
wait(), java.lang.Long*
wait(), java.lang.Object*
wait(), java.lang.String*
wait(), java.util.Date*
wait(), netscape.plugin.Plugin
watch(), Object object*
watch(), Object.watch()
write(), Document object*
write(), Document.write()
write(), File object*
write(), File object*
write(), File.write()
write(), Global object*
write(), response object*
write(), response.write()
Write(), TextStream object*
Write(), TextStream.Write()
WriteBlankLines(), TextStream object*
WriteBlankLines(),
TextStream.WriteBlankLines()
writeByte(), File object*
writeByte(), File.writeByte()
WriteLine(), TextStream object*
WriteLine(), TextStream.WriteLine()
writeln(), Document object*
writeln(), Document.writeln()
writeln(), File object*
writeln(), File object*
writeln(), File.writeln()

Method/internal

put()*

Method/Java

booleanValue(),
JavaObject.booleanValue()*
getClass(), JavaObject.getClass()*

Method/static

parse(), Date.parse()*
UTC(), Date.UTC()*
fromCharCode(), String.fromCharCode()*

MIME type

text/JavaScript*

Object model

ASP*
Browser
Document*
WSH

Object/browser

Background object*
EmbedArray object*
FormArray object*
FormElement object*
FormElementsArray object
FrameArray object
Frames object*
History object*
ImageArray object*
InputArray object*
LinkArray object*
MimeType object*
MimeTypeArray object*

Object/browser (continued)

Navigator object*
OptionsArray object*
Plugin object*
PluginArray object*
Rect object*
Screen object*
ScriptArray object*
Selection object*
SelectorArray object*
Window object*

Object/core

Arguments object*
Array object*
Boolean object*
Date object*
Error object*
EvalError object*
Function object*
Global object*
Math object*
Number object*
Object object*
RangeError object*
ReferenceError object*
RegExp object*
String object*
SyntaxError object*
TypeError object*
URIError object*

Object/CSS

style object (2)*

Object/DOM

AbstractView object*
AnchorArray object*
AppletArray object*
Attr object*
Attribute object*
Attributes object*
Button object*
CDATASection object*
CharacterData object*
Checkbox object*
ChildNodes object*
Collection object*
COMMENT object*
Doctype object*
DocumentEvent*
DocumentFragment object*
DocumentStyle object*
DocumentType object
DOMImplementation object
Entity object*
EntityReference object*
Event object*
EventException object*
EventListener object*
EventTarget object*
FileUpload object*
Frame object*
Hidden object*
Implementation object*
Input object*
LinkStyle object*
Location object*
MediaList object*
ModElement object*
MouseEvent object*
MutationEvent object*
NamedNodeMap object*
Node object*
NodeList object*
Notation object*
OptionElement object
Password object*
ProcessingInstruction object*
RadioButton object*
ResetButton object*
rule object*
StyleSheet object*
StyleSheetList object*
SubmitButton object*
TableSectionElement object
Text object
TEXTAREA object*
TextCell object*
textNode object*
UIEvent object*
userDefined object*

Object/HTML

! object*
<!-- ... --> (Comment block)
A object
ABBR object
ACRONYM object
ADDRESS object
Anchor object*
Applet object*
Area object*
B object
BASE object*
BASEFONT object*
BDO object*
BGSOUND object*
BIG object
BLOCKQUOTE object*
BODY object*
BR object*
BUTTON object*
CAPTION object*
CENTER object
CITE object
CODE object
COL object*
COLGROUP object*
DD object*
DEL object*
DFN object
DIR object*
DIV object*
DL object*
Document object*
DT object*
Element object*
EM object*
Embed object*
FIELDSET object*
FONT object*

Form object*
FRAMESET object*
H<n> object*
HEAD object*
HR object*
HTML object*
HyperLink object*
I object*
IFRAME object*
Image object*
IMG object*
INS object*
ISINDEX object*
KBD object*
Label object*
Legend object*
LI object*
LINK object*
LISTING object*
Map object*
MARQUEE object*
MENU object*
META object*
NOFRAMES object*
NOSCRIPT object*
OBJECT object*
OL object*
OptGroupElement object*
Option object*
P object*
ParamElement object*
PLAINTEXT object
PRE object*
Q object
RT object
RUBY object*
S object
SAMP object
SCRIPT object*
Select object*
SMALL object
SPAN object*
STRIKE object
STRONG object*
STYLE object (1)*
SUB object
SUP object
TABLE object*
TableColElement object*
TBODY object*
TD object*
TFOOT object*
TH object*
THEAD object*
TITLE object*
TR object*
TT object*
U object*
UL object*
Url object*
VAR object
XMP object

Object/internal

Activation object
Call object
Closure object*

Object/JScript

ActiveXObject object*
Automation object
clipboardData object*
currentStyle object*
dataTransfer object*
Dialog object*
Dictionary object*
Drive object*
Drives object*
Enumerator object*
external object*
File object*
Files object*
FileSystem object*
Filter object*
Filters object
Folder object*
Folders object*
runtimeStyle object*
TextRange object*
textRectangle object*
TextStream object*
userProfile object*
VBArray object*
vCard object*
XML object*

Object/JSS

JSSClasses object*
JSSTag object*
JSSTags object*

Object/Navigator

Bar object*
Clip object*
Closure()*
Crypto object*
EventCapturer object*
JavaArray object*
JavaClass object*
JavaMethod object
JavaObject object*
JavaPackage object*
Layer object*
LayerArray object*
Pkcs11 object*
Sidebar object*

Object/NES

blob object*
client object*
Connection object*
Cursor object*
database object*
DbPool object*
File object*
Lock object*
project object*
request object*
response object*
ResultSet object*
SendMail object*
server object*
Stproc object*

Object/WSH
WScript object*

Operator/additive
- (Minus)
+ (Add)
Add (+)*
Minus (-)
Subtract (-)*

Operator/assignment
%= (Modulo assign)
&= (Bitwise AND assign)
*= (Multiply assign)
/= (Divide assign)
^= (Bitwise XOR assign)
|= (Bitwise OR assign)
+= (Add assign)
<<= (Bitewise shift left assign)
= (Assign)*
-= (Minus assign)
>>= (Bitwise shift right assign)
>>>= (Bitwise unsigned shift right assign)
Add then assign (+=)*
Assign value (=)*
Bitwise AND then assign (&=)*
Bitwise OR then assign (|=)*
Bitwise shift left then assign (<<=)*
Bitwise shift right and assign (>>=)*
Bitwise unsigned shift right and assign (>>>=)*
Bitwise XOR and assign (^=)*
Concatenate then assign (+=)*
Divide then assign (/=)*
Minus then assign (-=)
Multiply then assign (*=)*
Remainder then assign (%=)*
Subtract then assign (-=)*

Operator/bitwise
& (Bitwise AND)
^ (Bitwise XOR)
| (Bitwise OR)
~ (Bitwise NOT)
<< (Bitwise shift left)
>> (Bitwise shift right)
>>> (Bitwise unsigned shift right)
Bitwise AND (&)*
Bitwise NOT - complement (~)*
Bitwise OR (|)*
Bitwise shift left (<<)*
Bitwise shift right (>>)*
Bitwise unsigned shift right (>>>)*
Bitwise XOR (^)*
Left shift
Right shift

Operator/conditional
?: (Conditional block)
Conditionally execute (?:)*

Operator/equality
!= (NOT equal)
== (Equal to)
Equal to (==)*
NOT Equal to (!=)*

Operator/identity
!== (NOT identical)
=== (Identical to)
Exactly equal to (===)
Identically equal to (===)*
NOT Identically equal to (!==)*
Strictly equal to (===)

Operator/internal
ToBoolean*
ToInt32*
ToInteger*
ToNumber*
ToObject*
ToPrimitive*
ToString*
ToUint16*
ToUint32*

Operator/logical
! (Logical NOT)
&& (Logical AND)
|| (Logical OR)
in*
instanceof*
Logical AND (&&)*
Logical NOT - complement (!)*
Logical OR (||)*
Logical XOR*

Operator/multiplicative
% (Modulo/remainder)
* (Multiply)
/ (Divide)
Divide (/)*
Modulo
Multiply (*)*
Remainder (%)*

Operator/postfix
-- (Post decrement)
++ (Post increment)
Decrement value (--)*
Increment value (++)*
Postfix decrement (--)*
Postfix expression*
Postfix increment (++)*

Operator/prefix
-- (Pre decrement)
++ (Pre increment)
Prefix decrement (--)*
Prefix expression*
Prefix increment (++)*

Operator/relational
< (Less than)
<= (Less than or equal to)
> (Greater than)
>= (Greater than or equal to)
Greater than (>)*
Greater than or equal to (>=)*
Less than (<)*
Less than or equal to (<=)*

Operator/string
+ (Concatenate)
Concatenate (+)
String concatenate (+)*

Operator/unary
- (Unary minus)
+ (Unary plus)
delete*
Negation operator (-)*
new*
Positive value (+)*
typeof*
void*

Overview (see also Background, Definition)
Character-case mapping
Compliance
JavaScript language
Lexical element
Pointers
Topic classification

Pitfall (see also Advice, Useful tip)
</SCRIPT>*
Bar.visibility*
Deprecated functionality*
Escaped JavaScript quotes in HTML*
Hiding scripts from old browsers*
HTML entity escape*
JavaScript entity*
Newlines are not <BR> tags*
Off by one errors*
style.zOrder*

Pre-processor
@*/*
@_alpha
@_jscript
@_jscript_build
@_jscript_version
@_mac
@_mc680x0
@_PowerPC
@_win16
@_win32
@_x86
@<variable_name>
@cc_on
@elif(...) ...
@else ...
@end
@if(...) ...
@set
Conditional code block*
Pre-processing - @_alpha*
Pre-processing - @_jscript*
Pre-processing - @_jscript_build*
Pre-processing - @_jscript_version*
Pre-processing - @_mac*
Pre-processing - @_mc680x0*
Pre-processing - @_PowerPC*
Pre-processing - @_win16*
Pre-processing - @_win32*
Pre-processing - @_x86*
Pre-processing - @<variable_name>*
Pre-processing - @cc_on*
Pre-processing - @elif(...) ...*
Pre-processing - @else ...*
Pre-processing - @end*
Pre-processing - @if(...) ...*
Pre-processing - @set*

Primitive value
Boolean literal*
Boolean*
false*
Null literal*
null*
Number*
Numeric literal*
String literal*
String*
true*

Product
Active Server Pages
ActiveX
ADO
ASP
fdlibm
IIS
Internet Information Server
LiveConnect
LiveScript
LiveWire
NES
Netscape Enterprise Server
Nombas ScriptEase
Perl Connect
ScriptEase
Windows Script Host
WSH

Property
!.tabIndex
$', RegExp object*
$&, RegExp object*
$*, RegExp object*
$_, RegExp object*
$_, RegExp.$_
$`, RegExp object*
$+, RegExp object*
__parent__
__parent__, Closure object*
__parent__, Closure.__parent__
__parent__, Object object*
__parent__, Object.__parent__
__proto__
__proto__, Closure object*
__proto__, Closure.__proto__
__proto__, Object object*
__proto__, Object.__proto__
<column_name>, Cursor object*
<column_name>, Cursor.<column_name>
<form_name>, Document object*
<form_name>, Document.<form_name>
<input_name>, request object*

Property (continued)

Property (continued)

Property (continued)

clip.right, Layer.clip.right
clip.right, style.clip.right
clip.top, Layer.clip.top
clip.top, style.clip.top
clip.width, Layer.clip.width
clipboardData
clipboardData, Global object*
clipboardData, Window object*
clipboardData, Window.clipboardData
cllientX, MouseEvent object*
closed
closed, Global object*
closed, Window object*
closed, Window.closed
code, Applet object*
code, Applet.code
code, EventException object*
code, EventException.code
code, OBJECT object*
code, OBJECT.code
codeBase, Applet object*
codeBase, Applet.codeBase
codeBase, OBJECT object*
codeBase, OBJECT.codeBase
codeType, OBJECT object*
codeType, OBJECT.codeType
color, BASEFONT object*
color, BASEFONT.color
Color, filter - Chroma()*
Color, filter - DropShadow()*
Color, filter - Glow()*
Color, filter - Mask()*
Color, filter - MaskFilter()*
Color, filter - Shadow()*
color, FONT object*
color, FONT.color
color, HR object*
color, HR.color
color, JSSTag object*
color, JSSTag.color
color, style object (2)*
color, style.color
colorDepth, Screen object*
colorDepth, Screen.colorDepth
colorProfile, style object (2)*
colorProfile, style.colorProfile
cols, FRAMESET object*
cols, FRAMESET.cols
cols, TABLE object*
cols, TABLE.cols
cols, TEXTAREA object*
cols, TEXTAREA.cols
colSpan, TD object*
colSpan, TD.colSpan
colSpan, TH object*
colSpan, TH.colSpan
Column, TextStream object*
Column, TextStream.Column
columnSpan, style object (2)*
columnSpan, style.columnSpan
compact, DIR object*
compact, DIR.compact
compact, DL object*
compact, DL.compact
compact, MENU object*
compact, MENU.compact
compact, OL object*
compact, OL.compact
compact, UL object*
compact, UL.compact
complete, Image object*
complete, Image.complete
complete, IMG object*
complete, IMG.complete
constructor, Array object*
constructor, Array.constructor
constructor, Boolean object*
constructor, Boolean.constructor
constructor, Crypto object*
constructor, Crypto.constructor
constructor, Date object*
constructor, Date.constructor
constructor, Enumerator object*
constructor, Enumerator.constructor
constructor, Error object*
constructor, Error.constructor
constructor, File object*
constructor, File.constructor
constructor, Function object*
constructor, Function.constructor
constructor, Image object*
constructor, Image.constructor
constructor, Lock object*
constructor, Lock.constructor
constructor, Math object*
constructor, Math.constructor
constructor, Navigator object*
constructor, Navigator.constructor
constructor, Number object*
constructor, Number.constructor
constructor, Object object*
constructor, Object.constructor
constructor, RegExp object*
constructor, RegExp.constructor
constructor, SendMail object*
constructor, SendMail.constructor
constructor, String object*
constructor, String.constructor
content, META object*
content, META.content
content, style object (2)*
content, style.content
contentEditable, Element object*
contentEditable, Element.contentEditable
cookie, Document object*
cookie, Document.cookie
cookieEnabled, Navigator object*
cookieEnabled, Navigator.cookieEnabled
coords, Anchor object*
coords, Anchor.coords
coords, Area object*
coords, Area.coords
coords, Url object*
coords, Url.coords
Count, Dictionary object*
Count, Dictionary.Count
count, Drives object*
Count, Files object*
Count, Files.Count
Count, Folders object*
Count, Folders.Count
counterIncrement, style object (2)*

Property (continued)

counterIncrement, style.counterIncrement
counterReset, style object (2)*
counterReset, style.counterReset
cpuClass, Navigator object*
cpuClass, Navigator.cpuClass
crypto
crypto, Global object*
crypto, Window object*
crypto, Window.crypto
cssFloat, style object (2)*
cssFloat, style.cssFloat
cssText, rule object*
cssText, rule.cssText
cssText, style object (2)*
cssText, style.cssText
cssText, StyleSheet object*
cssText, StyleSheet.cssText
ctrlKey, Event object*
ctrlKey, Event.ctrlKey
ctrlKey, MouseEvent object*
ctrlKey, MouseEvent.ctrlKey
cue, style object (2)*
cue, style.cue
cueAfter, style object (2)*
cueAfter, style.cueAfter
cueBefore, style object (2)*
cueBefore, style.cueBefore
current, History object*
current, History.current
currentStyle, Element object*
currentStyle, Element.currentStyle
currentTarget, Event object*
currentTarget, Event.currentTarget
currentTarget, MouseEvent object*
currentTarget, MutationEvent object*
cursor, style object (2)*
cursor, style.cursor
data, CharacterData object*
data, CharacterData.data
data, Event object*
data, Event.data
data, OBJECT object*
data, OBJECT.data
data, ProcessingInstruction object*
data, ProcessingInstruction.data
data, textNode object*
data, textNode.data
database, response object*
database, response.database
dataFld, A object
dataFld, Anchor object*
dataFld, Anchor.dataFld
dataFld, Applet object*
dataFld, BUTTON object*
dataFld, DIV object*
dataFld, Event object*
dataFld, File object*
dataFld, Frame object*
dataFld, IFRAME object*
dataFld, IMG object*
dataFld, Input object*
dataFld, Input.dataFld
dataFld, Label object*
dataFld, MARQUEE object*
dataFld, OBJECT object*
dataFld, Select object*
dataFld, SPAN object*
dataFld, TABLE object*
dataFormatAs, BUTTON object*
dataFormatAs, DIV object*
dataFormatAs, IMG object*
dataFormatAs, Input object*
dataFormatAs, Input.dataFormatAs
dataFormatAs, Label object*
dataFormatAs, MARQUEE object*
dataFormatAs, SPAN object*
dataPageSize, TABLE object*
dataPageSize, TABLE.dataPageSize
dataSrc, A object
dataSrc, Anchor object*
dataSrc, Anchor.dataSrc
dataSrc, Applet object*
dataSrc, BUTTON object*
dataSrc, DIV object*
dataSrc, File object*
dataSrc, Frame object*
dataSrc, IFRAME object*
dataSrc, IMG object*
dataSrc, Input object*
dataSrc, Input.dataSrc
dataSrc, Label object*
dataSrc, MARQUEE object*
dataSrc, OBJECT object*
dataSrc, Select object*
dataSrc, SPAN object*
dataSrc, TABLE object*
dataTransfer, Event object*
dataTransfer, Event.dataTransfer
DateCreated, File object*
DateCreated, File.DateCreated
DateCreated, Folder object*
DateCreated, Folder.DateCreated
DateLastAccessed, File object*
DateLastAccessed, File.DateLastAccessed
DateLastAccessed, Folder object*
DateLastAccessed,
Folder.DateLastAccessed
DateLastModified, File object*
DateLastModified, File.DateLastModified
DateLastModified, Folder object*
DateLastModified, Folder.DateLastModified
dateTime, DEL object*
dateTime, DEL.dateTime
dateTime, INS object*
dateTime, INS.dateTime
dateTime, ModElement object*
dateTime, ModElement.dateTime
declare, OBJECT object*
declare, OBJECT.declare
defaultCharset, Document object*
defaultCharset, Document.defaultCharset
defaultChecked, Checkbox object*
defaultChecked, Checkbox.defaultChecked
defaultChecked, Input object*
defaultChecked, Input.defaultChecked
defaultChecked, RadioButton object*
defaultChecked,
RadioButton.defaultChecked
defaultSelected, Input object*
defaultSelected, Option object*
defaultSelected, Option.defaultSelected
defaultStatus
defaultStatus, Frame object*

Property (continued)

defaultStatus, Frame.defaultStatus
defaultStatus, Global object*
defaultStatus, Window object*
defaultStatus, Window.defaultStatus
defaultValue, File object*
defaultValue, Image object*
defaultValue, IMG object*
defaultValue, Input object*
defaultValue, Input.defaultValue
defer, SCRIPT object*
defer, SCRIPT.defer
defer, XML object*
defer, XML.defer
description, Error object*
description, Error.description
description, JavaObject object*
description, MimeType object*
description, MimeType.description
description, Plugin object*
description, Plugin.description
description, URIError object*
designMode, Document object*
designMode, Document.designMode
detail, MouseEvent object*
detail, UIEvent object*
detail, UIEvent.detail
dialogArguments
dialogArguments, Global object*
dialogArguments, Window object*
dialogArguments, Window.dialogArguments
dialogHeight
dialogHeight, Global object*
dialogHeight, Window object*
dialogHeight, Window.dialogHeight
dialogLeft
dialogLeft, Global object*
dialogLeft, Window object*
dialogLeft, Window.dialogLeft
dialogTop
dialogTop, Global object*
dialogTop, Window object*
dialogTop, Window.dialogTop
dialogWidth
dialogWidth, Global object*
dialogWidth, Window object*
dialogWidth, Window.dialogWidth
dir, BDO object*
dir, BDO.dir
dir, Element object*
dir, Element.dir
dir, NOFRAMES object*
dir, NOFRAMES.dir
dir, NOSCRIPT object*
dir, NOSCRIPT.dir
Direction, filter - Blinds()*
Direction, filter - CheckerBoard()*
Direction, filter - MotionBlur()*
Direction, filter - Shadow()*
direction, MARQUEE object*
direction, MARQUEE.direction
direction, style object (2)*
direction, style.direction
disabled, Input object*
disabled, Input.disabled
disabled, LINK object*
disabled, LINK.disabled
disabled, OptGroupElement object*
disabled, OptGroupElement.disabled
disabled, STYLE object (1)*
disabled, STYLE.disabled
disabled, StyleSheet object*
disabled, StyleSheet.disabled
display, JSSTag object*
display, JSSTag.display
display, style object (2)*
display, style.display
doctype, Document object*
doctype, Document.doctype
document
document, Element object*
document, Element.document
document, Global object*
document, Layer object*
document, Layer.document
document, Window object*
document, Window.document
documentElement, Document object*
documentElement,
Document.documentElement
domain, Document object*
domain, Document.domain
Drive, File object*
Drive, File.Drive
Drive, Folder object*
Drive, Folder.Drive
DriveLetter, Drive object*
DriveLetter, Drive.DriveLetter
DriveType, Drive object*
DriveType, Drive.DriveType
dropEffect, dataTransfer object*
dropEffect, dataTransfer.dropEffect
Duration, filter - Barn()*
Duration, filter - Blinds()*
Duration, filter - CheckerBoard()*
Duration, filter - Fade()*
Duration, filter - GradientWipe()*
Duration, filter - Inset()*
Duration, filter - Iris()*
Duration, filter - Pixelate()*
Duration, filter - Pixelate()*
Duration, filter - RadialWipe()*
Duration, filter - RandomBars()*
Duration, filter - RandomDissolve()*
Duration, filter - Slide()*
Duration, filter - Spiral()*
Duration, filter - Stretch()*
Duration, filter - Strips()*
Duration, filter - Wheel()*
Duration, filter - Zigzag()*
Dx, filter - Matrix()*
Dy, filter - Matrix()*
dynsrc, IMG object*
dynsrc, IMG.dynsrc
effectAllowed, dataTransfer object*
effectAllowed, dataTransfer.effectAllowed
elements, Form object*
elements.length, Form.elements.length
elevation, style object (2)*
elevation, style.elevation
emptyCells, style object (2)*
emptyCells, style.emptyCells
Enabled, filter - Alpha()*
Enabled, filter - AlphaImageLoader()*

Enabled, filter - Barn()*
Enabled, filter - BasicImage()*
Enabled, filter - Blinds()*
Enabled, filter - Blur()*
Enabled, filter - CheckerBoard()*
Enabled, filter - Chroma()*
Enabled, filter - Compositor()*
Enabled, filter - DropShadow()*
Enabled, filter - Emboss()*
Enabled, filter - Engrave()*
Enabled, filter - Fade()*
Enabled, filter - Glow()*
Enabled, filter - Gradient()*
Enabled, filter - GradientWipe()*
Enabled, filter - Inset()*
Enabled, filter - Iris()*
Enabled, filter - Light()*
Enabled, filter - Matrix()*
Enabled, filter - MotionBlur()*
Enabled, filter - Pixelate()*
Enabled, filter - Pixelate()*
Enabled, filter - RadialWipe()*
Enabled, filter - RandomBars()*
Enabled, filter - RandomDissolve()*
Enabled, filter - Slide()*
Enabled, filter - Spiral()*
Enabled, filter - Stretch()*
Enabled, filter - Strips()*
Enabled, filter - Wave()*
Enabled, filter - Wheel()*
Enabled, filter - Zigzag()*
enabled, Filter object*
enabled, Filter.enabled
Enabled , filter - MaskFilter()*
Enabled , filter - Shadow()*
enabledPlugin, MimeType object*
enabledPlugin, MimeType.enabledPlugin
encoding, Form object*
encoding, Form.encoding
enctype, Form object*
enctype, Form.enctype
EndColor, filter - Gradient()*
EndColorStr, filter - Gradient()*
ErrorsTo, SendMail object*
ErrorsTo, SendMail.ErrorsTo
event
event, Global object*
event, SCRIPT object*
event, SCRIPT.event
event, Window object*
event, Window.event
event, XML object*
event, XML.event
eventPhase, Event object*
eventPhase, Event.eventPhase
eventPhase, MouseEvent object*
eventPhase, MutationEvent object*
expando, Document object*
expando, Document.expando
external
external, Global object*
external, Window object*
external, Window.external
face, BASEFONT object*
face, BASEFONT.face
face, FONT object*
face, FONT.face
fgColor, Document object*
fgColor, Document.fgColor
fileCreatedDate, Document object*
fileCreatedDate, Document.fileCreatedDate
fileCreatedDate, IMG object*
fileCreatedDate, IMG.fileCreatedDate
fileModifiedDate, Document object*
fileModifiedDate,
Document.fileModifiedDate
fileModifiedDate, IMG object*
fileModifiedDate, IMG.fileModifiedDate
filename, JavaObject object*
filename, Plugin object*
filename, Plugin.filename
fileSize, Document object*
fileSize, Document.fileSize
fileSize, IMG object*
fileSize, IMG.fileSize
FileSystem, Drive object*
FileSystem, Drive.FileSystem
fileUpdatedDate, IMG object*
fileUpdatedDate, IMG.fileUpdatedDate
filter, style object (2)*
filter, style.filter
FilterType, filter - Matrix()*
FinishOpacity, filter - Alpha()*
FinishX, filter - Alpha()*
FinishY, filter - Alpha()*
firstChild, Element object*
firstChild, Element.firstChild
firstChild, Node object*
firstChild, Node.firstChild
float, style object (2)*
float, style.float
floatStyle, style object (2)*
floatStyle, style.floatStyle
font, style object (2)*
font, style.font
fontFamily, JSSTag object*
fontFamily, JSSTag.fontFamily
fontFamily, style object (2)*
fontFamily, style.fontFamily
fontSize, JSSTag object*
fontSize, JSSTag.fontSize
fontSize, style object (2)*
fontSize, style.fontSize
fontSizeAdjust, style object (2)*
fontSizeAdjust, style.fontSizeAdjust
fontSmoothingEnabled, Screen object*
fontSmoothingEnabled,
Screen.fontSmoothingEnabled
fontStretch, style object (2)*
fontStretch, style.fontStretch
fontStyle, JSSTag object*
fontStyle, JSSTag.fontStyle
fontStyle, style object (2)*
fontStyle, style.fontStyle
fontVariant, style object (2)*
fontVariant, style.fontVariant
fontWeight, JSSTag object*
fontWeight, JSSTag.fontWeight
fontWeight, style object (2)*
fontWeight, style.fontWeight
form, Applet object*
form, BUTTON object*
form, FIELDSET object*
form, Input object*

Property (continued)

form, Input.form
form, ISINDEX object*
form, ISINDEX.form
form, Label object*
form, Legend object*
form, OBJECT object*
form, OBJECT.form
form, Option object*
form, Select object*
frame
frame, Global object*
frame, TABLE object*
frame, TABLE.frame
frame, Window object*
frame, Window.frame
frameBorder, Frame object*
frameBorder, Frame.frameBorder
frameBorder, FRAMESET object*
frameBorder, FRAMESET.frameBorder
frameBorder, IFRAME object*
frameBorder, IFRAME.frameBorder
frameRate
frameRate, Global object*
frameRate, Window object*
frameRate, Window.frameRate
frameSpacing, FRAMESET object*
frameSpacing, FRAMESET.frameSpacing
frameSpacing, IFRAME object*
frameSpacing, IFRAME.frameSpacing
FreeSpace, Drive object*
FreeSpace, Drive.FreeSpace
Freq, filter - Wave()*
From, SendMail object*
From, SendMail.From
fromElement, Event object*
fromElement, Event.fromElement
FullName, WScript object*
FullName, WScript.FullName
Function, filter - Compositor()*
global, RegExp object*
global, RegExp.global
GradientSize, filter - GradientWipe()*
GradientType, filter - Gradient()*
GrayScale, filter - BasicImage()*
GridSizeX, filter - Spiral()*
GridSizeX, filter - Zigzag()*
GridSizeY, filter - Spiral()*
GridSizeY, filter - Zigzag()*
hash, A object
hash, Anchor object*
hash, Anchor.hash
hash, Area object*
hash, Area.hash
hash, Location object*
hash, Location.hash
hash, Url object*
hash, Url.hash
headers, TD object*
headers, TD.headers
headers, TH object*
headers, TH.headers
height, Applet object*
height, Applet.height
height, Clip object*
height, Clip.height
height, Document object*
height, Document.height
height, Embed object*
height, Embed.height
height, Event object*
height, Event.height
height, Frame object*
height, Frame.height
height, IFRAME object*
height, IFRAME.height
height, Image object*
height, Image.height
height, IMG object*
height, IMG.height
height, JSSTag object*
height, JSSTag.height
height, MARQUEE object*
height, MARQUEE.height
height, OBJECT object*
height, OBJECT.height
height, Rect object*
height, Rect.height
height, Screen object*
height, Screen.height
height, style object (2)*
height, style.height
height, TABLE object*
height, TABLE.height
height, TD object*
height, TD.height
height, TH object*
height, TH.height
hidden, Embed object*
hidden, Embed.hidden
hidden, Layer object*
hidden, Layer.hidden
hideFocus, Element object*
hideFocus, Element.hideFocus
history
history, Global object*
history, Window object*
history, Window.history
host, A object
host, Anchor object*
host, Anchor.host
host, Area object*
host, Area.host
host, Location object*
host, Location.host
host, server object*
host, server.host
host, Url object*
host, Url.host
hostname, A object
hostname, Anchor object*
hostname, Anchor.hostname
hostname, Area object*
hostname, Area.hostname
hostname, Location object*
hostname, Location.hostname
hostname, server object*
hostname, server.hostname
hostname, Url object*
hostname, Url.hostname
href, A object
href, Anchor object*
href, Anchor.href
href, Area object*

Property (continued)

layoutGridCharSpacing, style object (2)*
layoutGridCharSpacing, style.layoutGridCharSpacing
layoutGridLine, style object (2)*
layoutGridLine, style.layoutGridLine
layoutGridMode, style object (2)*
layoutGridMode, style.layoutGridMode
layoutGridType, style object (2)*
layoutGridType, style.layoutGridType
left, Clip object*
left, Clip.left
left, JSSTag object*
left, Layer object*
left, Layer.left
left, Rect object*
left, Rect.left
left, style object (2)*
left, style.left
left, textRectangle object*
left, textRectangle.left
leftMargin, BODY object*
leftMargin, BODY.leftMargin
length
length, AnchorArray object*
length, AnchorArray.length
length, AppletArray object*
length, AppletArray.length
length, Arguments object*
length, Arguments.length
length, Array object*
length, Array.length
length, Attributes object*
length, Attributes.length
length, CharacterData object*
length, CharacterData.length
length, Collection object*
length, Collection.length
length, Date object*
length, Date.length
length, EmbedArray object*
length, EmbedArray.length
length, Filters object
length, Filters.length
length, Form object*
length, Form.length
length, FormArray object*
length, FormArray.length
length, FormElementsArray object
length, FormElementsArray.length
length, FrameArray object
length, FrameArray.length
length, Frames object*
length, Frames.length
length, Function object*
length, Function.length
length, Global object*
length, History object*
length, History.length
length, ImageArray object*
length, ImageArray.length
length, Input object*
length, InputArray object*
length, JavaArray object*
length, JavaArray.length
length, JavaObject object*
length, LayerArray object*
length, LayerArray.length
length, LinkArray object*
length, LinkArray.length
length, MimeTypeArray object*
length, MimeTypeArray.length
length, NamedNodeMap object*
length, NamedNodeMap.length
length, NodeList object*
length, NodeList.length
length, OptionsArray object*
length, OptionsArray.length
length, Plugin object*
length, Plugin.length
length, PluginArray object*
length, PluginArray.length
length, rows object
length, ScriptArray object*
length, ScriptArray.length
length, Select object*
length, Select.length
length, SelectorArray object*
length, SelectorArray.length
length, String object*
length, String.length
length, style object (2)*
length, style.length
length, StyleSheetList object*
length, StyleSheetList.length
length, textNode object*
length, textNode.length
length, Window object*
length, Window.length
letterSpacing, style object (2)*
letterSpacing, style.letterSpacing
LightStrength, filter - Wave()*
Line, TextStream object*
Line, TextStream.Line
lineBreak, style object (2)*
lineBreak, style.lineBreak
lineHeight, JSSTag object*
lineHeight, JSSTag.lineHeight
lineHeight, style object (2)*
lineHeight, style.lineHeight
link, BODY object*
link, BODY.link
linkColor, Document object*
linkColor, Document.linkColor
listStyle, style object (2)*
listStyle, style.listStyle
listStyleImage, style object (2)*
listStyleImage, style.listStyleImage
listStylePosition, style object (2)*
listStylePosition, style.listStylePosition
listStyleType, JSSTag object*
listStyleType, JSSTag.listStyleType
listStyleType, style object (2)*
listStyleType, style.listStyleType
location
location, Document object*
location, Document.location
location, Global object*
location, Window object*
location, Window.location
locationbar
locationbar, Global object*
locationbar, Window object*
locationbar, Window.locationbar

longDesc, Frame object*
longDesc, Frame.longDesc
longDesc, IFRAME object*
longDesc, IFRAME.longDesc
longDesc, IMG object*
longDesc, IMG.longDesc
loop, BGSOUND object*
loop, BGSOUND.loop
loop, IMG object*
loop, IMG.loop
loop, MARQUEE object*
loop, MARQUEE.loop
lowsrc, Image object*
lowsrc, Image.lowsrc
lowsrc, IMG object*
lowsrc, IMG.lowsrc
M11, filter - Matrix()*
M12, filter - Matrix()*
M21, filter - Matrix()*
M22, filter - Matrix()*
MakeShadow, filter - Blur()*
margin, FIELDSET object*
margin, FIELDSET.margin
margin, style object (2)*
margin, style.margin
marginBottom, JSSTag object*
marginBottom, JSSTag.marginBottom
marginBottom, style object (2)*
marginBottom, style.marginBottom
marginHeight, Frame object*
marginHeight, Frame.marginHeight
marginHeight, IFRAME object*
marginHeight, IFRAME.marginHeight
marginLeft, JSSTag object*
marginLeft, JSSTag.marginLeft
marginLeft, style object (2)*
marginLeft, style.marginLeft
marginRight, JSSTag object*
marginRight, JSSTag.marginRight
marginRight, style object (2)*
marginRight, style.marginRight
marginTop, JSSTag object*
marginTop, JSSTag.marginTop
marginTop, style object (2)*
marginTop, style.marginTop
marginWidth, Frame object*
marginWidth, Frame.marginWidth
marginWidth, IFRAME object*
marginWidth, IFRAME.marginWidth
markerOffset, style object (2)*
markerOffset, style.markerOffset
marks, style object (2)*
marks, style.marks
Mask, filter - BasicImage()*
MaskColor, filter - BasicImage()*
Math, Global object*
Math, Window object*
maxHeight, style object (2)*
maxHeight, style.maxHeight
maxLength, Input object*
maxLength, Input.maxLength
maxLength, Password object*
maxLength, Password.maxLength
maxLength, TextCell object*
maxLength, TextCell.maxLength
MaxSquare, filter - Pixelate()*
MaxSquare, filter - Pixelate()*
maxWidth, style object (2)*
maxWidth, style.maxWidth
media, LINK object*
media, LINK.media
media, STYLE object (1)*
media, STYLE.media
media, StyleSheet object*
media, StyleSheet.media
menuArguments, external object*
menuArguments, external.menuArguments
menubar
menubar, Global object*
menubar, Window object*
menubar, Window.menubar
message, Error object*
message, Error.message
message, URIError object*
metaKey, MouseEvent object*
metaKey, MouseEvent.metaKey
method, Form object*
method, Form.method
method, request object*
method, request.method
Methods, A object
Methods, Anchor object*
Methods, Anchor.Methods
Methods, Url object*
Methods, Url.Methods
mimeType, A object
mimeType, Anchor object*
mimeType, Anchor.mimeType
mimeType, Url object*
mimeType, Url.mimeType
minHeight, style object (2)*
minHeight, style.minHeight
minWidth, style object (2)*
minWidth, style.minWidth
Mirror, filter - BasicImage()*
modifiers, Event object*
modifiers, Event.modifiers
Motion, filter - Barn()*
Motion, filter - GradientWipe()*
Motion, filter - Iris()*
Motion, filter - Strips()*
multiple, Select object*
multiple, Select.multiple
name
name, Anchor object*
name, Anchor.name
name, Applet object*
name, Applet.name
name, Area object*
name, Area.name
name, Attribute object*
name, Attribute.name
name, BUTTON object*
name, BUTTON.name
name, Doctype object*
name, Doctype.name
name, Embed object*
name, Embed.name
name, Error object*
Name, Error.name
Name, File object*
Name, File.Name
Name, Folder object*
name, Folder.Name

Property (continued)

name, Form object*
name, Form.name
name, Frame object*
name, Frame.name
name, Global object*
name, IFRAME object*
name, IFRAME.name
name, Image object*
name, Image.name
name, IMG object*
name, IMG.name
name, Input object*
name, Input.name
name, JavaObject object*
name, Layer object*
name, Layer.name
name, Map object*
name, Map.name
name, META object*
name, META.name
name, MimeType object*
name, MimeType.name
name, Object object*
name, OBJECT object*
name, Object.name
name, OBJECT.name
name, ParamElement object*
name, ParamElement.name
name, Plugin object*
name, Plugin.name
name, URIError object*
name, Url object*
name, Url.name
name, Window object*
Name, Window.name
Name, WScript object*
name, WScript.Name
nameProp, A object
nameProp, Anchor object*
nameProp, Anchor.nameProp
nameProp, Url object*
nameProp, Url.nameProp
navigator
navigator, Global object*
navigator, Window object*
navigator, Window.navigator
netscape, Global object*
netscape, Window object*
netscape, Window.netscape
Network, WScript object*
Network, WScript.Network
newValue, MutationEvent object*
newValue, MutationEvent.newValue
next, History object*
next, History.next
nextSibling, Element object*
nextSibling, Element.nextSibling
nextSibling, Node object*
nextSibling, Node.nextSibling
nodeName, Attribute object*
nodeName, Attribute.nodeName
nodeName, Element object*
nodeName, Element.nodeName
nodeName, Node object*
nodeName, Node.nodeName
nodeType, Attribute object*
nodeType, Attribute.nodeType
nodeType, Element object*
nodeType, Element.nodeType
nodeType, Node object*
nodeType, Node.nodeType
nodeValue, Attribute object*
nodeValue, Attribute.nodeValue
nodeValue, Element object*
nodeValue, Element.nodeValue
nodeValue, Node object*
nodeValue, Node.nodeValue
noHref, Area object*
noHref, Area.noHref
noResize, Frame object*
noResize, Frame.noResize
noResize, IFRAME object*
noResize, IFRAME.noResize
noShade, HR object*
noShade, HR.noShade
notationName, Entity object*
notationName, Entity.notationName
noWrap, BODY object*
noWrap, BODY.noWrap
noWrap, DD object*
noWrap, DD.noWrap
noWrap, DT object*
noWrap, DT.noWrap
noWrap, TD object*
noWrap, TD.noWrap
noWrap, TH object*
noWrap, TH.noWrap
number, Error object*
number, Error.number
number, URIError object*
object, Applet object*
object, Applet.object
object, OBJECT object*
object, OBJECT.object
offScreenBuffering
offScreenBuffering, Global object*
offscreenBuffering, Window object*
offscreenBuffering,
Window.offscreenBuffering
offsetHeight, Element object*
offsetHeight, Element.offsetHeight
offsetLeft, Element object*
offsetLeft, Element.offsetLeft
offsetParent, Element object*
offsetParent, Element.offsetParent
offsetTop, Element object*
offsetTop, Element.offsetTop
offsetWidth, Element object*
offsetWidth, Element.offsetWidth
offsetX, Event object*
offsetX, Event.offsetX
offsetY, Event object*
offsetY, Event.offsetY
OffX, filter - DropShadow()*
OffY, filter - DropShadow()*
onblur, Window.onblur
ondragdrop, Window.ondragdrop
onerror, Window.onerror
onevent, Element.onevent
onevent, Input.onevent
onfocus, Window.onfocus
onLine, Navigator object*
onLine, Navigator.onLine

Property (continued)

parentWindow, Document.parentWindow
Path, Drive object*
Path, Drive.Path
Path, File object*
Path, File.Path
Path, Folder object*
Path, Folder.Path
Path, WScript object*
Path, WScript.Path
pathname, A object
pathname, Anchor object*
pathname, Anchor.pathname
pathname, Area object*
pathname, Area.pathname
pathname, Location object*
pathname, Location.pathname
pathname, Url object*
pathname, Url.pathname
pause, style object (2)*
pause, style.pause
pauseAfter, style object (2)*
pauseAfter, style.pauseAfter
pauseBefore, style object (2)*
pauseBefore, style.pauseBefore
Percent, filter - Barn()*
Percent, filter - Blinds()*
Percent, filter - Fade()*
Percent, filter - GradientWipe()*
Percent, filter - Inset()*
Percent, filter - Iris()*
Percent, filter - Pixelate()*
Percent, filter - Pixelate()*
Percent, filter - RadialWipe()*
Percent, filter - RandomBars()*
Percent, filter - RandomDissolve()*
Percent, filter - Slide()*
Percent, filter - Spiral()*
Percent, filter - Stretch()*
Percent, filter - Strips()*
Percent, filter - Wheel()*
Percent, filter - Zigzag()*
personalbar
personalbar, Global object*
personalbar, Window object*
personalbar, Window.personalbar
Phase, filter - Wave()*
pitch, style object (2)*
pitch, style.pitch
pitchRange, style object (2)*
pitchRange, style.pitchRange
pixelBottom, style object (2)*
pixelBottom, style.pixelBottom
pixelDepth, Screen object*
pixelDepth, Screen.pixelDepth
pixelHeight, style object (2)*
pixelHeight, style.pixelHeight
pixelLeft, style object (2)*
pixelLeft, style.pixelLeft
PixelRadius, filter - Blur()*
pixelRight, style object (2)*
pixelRight, style.pixelRight
pixelTop, style object (2)*
pixelTop, style.pixelTop
pixelWidth, style object (2)*
pixelWidth, style.pixelWidth
pkcs11, Global object*
pkcs11, Window object*
pkcs11, Window.pkcs11
platform, Navigator object*
platform, Navigator.platform
playDuring, style object (2)*
playDuring, style.playDuring
pluginspage, Embed object*
pluginspage, Embed.pluginspage
port, A object
port, Anchor object*
port, Anchor.port
port, Area object*
port, Area.port
port, Location object*
port, Location.port
port, server object*
port, server.port
port, Url object*
port, Url.port
posBottom, style object (2)*
posBottom, style.posBottom
posHeight, style object (2)*
posHeight, style.posHeight
position, style object (2)*
position, style.position
Positive , filter - DropShadow()*
posLeft, style object (2)*
posLeft, style.posLeft
posRight, style object (2)*
posRight, style.posRight
posTop, style object (2)*
posTop, style.posTop
posWidth, style object (2)*
posWidth, style.posWidth
previous, History object*
previous, History.previous
previousSibling, Element object*
previousSibling, Element.previousSibling
previousSibling, Node object*
previousSibling, Node.previousSibling
prevValue, MutationEvent object*
prevValue, MutationEvent.prevValue
profile, HEAD object*
profile, HEAD.profile
project, response object*
project, response.project
prompt, ISINDEX object*
prompt, ISINDEX object*
prompt, ISINDEX.prompt
propertyName, Event object*
propertyName, Event.propertyName
protocol, A object
protocol, Anchor object*
protocol, Anchor.protocol
protocol, Area object*
protocol, Area.protocol
protocol, Document object*
protocol, Document.protocol
protocol, IMG object*
protocol, IMG.protocol
protocol, Location object*
protocol, Location.protocol
protocol, request object*
protocol, request.protocol
protocol, server object*
protocol, server.protocol
protocol, Url object*

Property (continued)

right, style.right
right, textRectangle object*
right, textRectangle.right
rightMargin, BODY object*
rightMargin, BODY.rightMargin
RootFolder, Drive object*
RootFolder, Drive.RootFolder
Rotation, filter - BasicImage()*
rowIndex, TR object*
rowIndex, TR.rowIndex
rows, FRAMESET object*
rows, FRAMESET.rows
rows, TBODY object*
rows, TEXTAREA object*
rows, TEXTAREA.rows
rowSpan, style object (2)*
rowSpan, style.rowSpan
rowSpan, TD object*
rowSpan, TD.rowSpan
rowSpan, TH object*
rowSpan, TH.rowSpan
rubyAlign, style object (2)*
rubyAlign, style.rubyAlign
rubyOverhang, style object (2)*
rubyOverhang, style.rubyOverhang
rubyPosition, style object (2)*
rubyPosition, style.rubyPosition
rules, TABLE object*
rules, TABLE.rules
runtimeStyle, Element object*
runtimeStyle, Element.runtimeStyle
runtimeStyle, rule object*
runtimeStyle, rule.runtimeStyle
scheme, META object*
scheme, META.scheme
scope, TD object*
scope, TD.scope
scope, TH object*
scope, TH.scope
scopeName, Element object*
scopeName, Element.scopeName
screen
screen, Global object*
screen, Window object*
screen, Window.screen
screenLeft
screenLeft, Global object*
screenLeft, Window object*
screenLeft, Window.screenLeft
screenTop
screenTop, Global object*
screenTop, Window object*
screenTop, Window.screenTop
screenX
screenX, Event object*
screenX, Event.screenX
screenX, Global object*
screenX, MouseEvent object*
screenX, MouseEvent.screenX
screenX, Window object*
screenX, Window.screenX
screenY
screenY, Event object*
screenY, Event.screenY
screenY, Global object*
screenY, MouseEvent object*
screenY, MouseEvent.screenY
screenY, Window object*
screenY, Window.screenY
ScriptFullName, WScript object*
ScriptFullName, WScript.ScriptFullName
ScriptName, WScript object*
ScriptName, WScript.ScriptName
scroll, BODY object*
scroll, BODY.scroll
scrollAmount, MARQUEE object*
scrollAmount, MARQUEE.scrollAmount
scrollbar3dLightColor, style object (2)*
scrollbar3dLightColor, style.scrollbar3dLightColor
scrollbarArrowColor, style object (2)*
scrollbarArrowColor, style.scrollbarArrowColor
scrollbarBaseColor, style object (2)*
scrollbarBaseColor, style.scrollbarBaseColor
scrollbarDarkShadowColor, style object (2)*
scrollbarDarkShadowColor, style.scrollbarDarkShadowColor
scrollbarFaceColor, style object (2)*
scrollbarFaceColor, style.scrollbarFaceColor
scrollbarHighlightColor, style object (2)*
scrollbarHighlightColor, style.scrollbarHighlightColor
scrollbars
scrollbars, Global object*
scrollbars, Window object*
scrollbars, Window.scrollbars
scrollbarShadowColor, style object (2)*
scrollbarShadowColor, style.scrollbarShadowColor
scrollDelay, MARQUEE object*
scrollDelay, MARQUEE.scrollDelay
scrollHeight, Element object*
scrollHeight, Element.scrollHeight
scrolling, Frame object*
scrolling, Frame.scrolling
scrolling, IFRAME object*
scrolling, IFRAME.scrolling
scrollLeft, Element object*
scrollLeft, Element.scrollLeft
scrollTop, Element object*
scrollTop, Element.scrollTop
scrollWidth, Element object*
scrollWidth, Element.scrollWidth
search, A object
search, Anchor object*
search, Anchor.search
search, Area object*
search, Area.search
search, Location object*
search, Location.search
search, Url object*
search, Url.search
sectionRowIndex, TR object*
sectionRowIndex, TR.sectionRowIndex
secure
secure, Global object*
secure, Window object*
secure, Window.secure
securityPolicy, Navigator object*

securityPolicy, Navigator.securityPolicy
selected, Input object*
selected, Option object*
selected, Option.selected
selectedIndex, Input object*
selectedIndex, Select object*
selectedIndex, Select.selectedIndex
selection, Document object*
selection, Document.selection
selectorText, rule object*
selectorText, rule.selectorText
self
self, Global object*
self, Window object*
self, Window.self
SerialNumber, Drive object*
SerialNumber, Drive.SerialNumber
server, response object*
server, response.server
ShadowOpacity, filter - Blur()*
shape, Anchor object*
shape, Anchor.shape
shape, Area object*
shape, Area.shape
shape, Url object*
shape, Url.shape
ShareName, Drive object*
ShareName, Drive.ShareName
shiftKey, Event object*
shiftKey, Event.shiftKey
shiftKey, MouseEvent object*
shiftKey, MouseEvent.shiftKey
ShortName, File object*
ShortName, File.ShortName
ShortName, Folder object*
ShortName, Folder.ShortName
ShortPath, File object*
ShortPath, File.ShortPath
ShortPath, Folder object*
ShortPath, Folder.ShortPath
siblingAbove, Layer object*
siblingAbove, Layer.siblingAbove
siblingBelow, Layer object*
siblingBelow, Layer.siblingBelow
sidebar, Window object*
sidebar, Window.sidebar
size, BASEFONT object*
size, BASEFONT.size
Size, File object*
Size, File.Size
size, FileUpload object*
size, FileUpload.size
Size, Folder object*
Size, Folder.Size
size, FONT object*
size, FONT.size
size, HR object*
size, HR.size
size, Image object*
size, IMG object*
size, Input object*
size, Input.size
size, Password object*
size, Password.size
size, Select object*
size, Select.size
size, style object (2)*
size, style.size
size, TextCell object*
size, TextCell.size
SizingMethod, filter - AlphaImageLoader()*
SizingMethod, filter - Matrix()*
SlideStyle, filter - Slide()*
Smtpserver, SendMail object*
Smtpserver, SendMail.Smtpserver
source, RegExp object*
source, RegExp.source
sourceIndex, Element object*
sourceIndex, Element.sourceIndex
sourceIndex, Frame object*
span, COL object*
span, COL.span
span, COLGROUP object*
span, COLGROUP.span
span, TableColElement object*
span, TableColElement.span
speak, style object (2)*
speak, style.speak
speakDate, style object (2)*
speakDate, style.speakDate
speakHeader, style object (2)*
speakHeader, style.speakHeader
speakNumeral, style object (2)*
speakNumeral, style.speakNumeral
speakPunctuation, style object (2)*
speakPunctuation, style.speakPunctuation
speakTime, style object (2)*
speakTime, style.speakTime
specified, Attribute object*
specified, Attribute.specified
speechRate, style object (2)*
speechRate, style.speechRate
spokes, filter - Wheel()*
SquaresX, filter - CheckerBoard()*
SquaresY, filter - CheckerBoard()*
src, Applet object*
src, Applet.src
src, Background object*
src, Background.src
src, BGSOUND object*
src, BGSOUND.src
src, Embed object*
src, Embed.src
Src, filter - AlphaImageLoader()*
src, Frame object*
src, Frame.src
src, IFRAME object*
src, IFRAME.src
src, Image object*
src, Image.src
src, IMG object*
src, IMG.src
src, Input object*
src, Input.src
src, Layer object*
src, Layer.src
src, SCRIPT object*
src, SCRIPT.src
src, XML object*
src, XML.src
srcElement, Event object*
srcElement, Event.srcElement
srcFilter, Event object*
srcFilter, Event.srcFilter

Property (continued)

srcUrn, Event object*
standby, OBJECT object*
standby, OBJECT.standby
start, IMG object*
start, IMG.start
start, OL object*
start, OL.start
start, UL object*
StartColor, filter - Gradient()*
StartColorStr, filter - Gradient()*
StartX, filter - Alpha()*
StartY, filter - Alpha()*
status
status, BUTTON object*
status, Checkbox object*
status, Checkbox.status
status, filter - Barn()*
status, filter - Blinds()*
status, filter - Fade()*
status, filter - GradientWipe()*
status, filter - Inset()*
status, filter - Iris()*
status, filter - Pixelate()*
status, filter - Pixelate()*
status, filter - RadialWipe()*
status, filter - RandomBars()*
status, filter - RandomDissolve()*
status, filter - Slide()*
status, filter - Spiral()*
status, filter - Stretch()*
status, filter - Strips()*
status, filter - Wheel()*
status, filter - Zigzag()*
status, Global object*
status, Input object*
status, RadioButton object*
status, RadioButton.status
status, Window object*
status, Window.status
statusbar
statusbar, Global object*
statusbar, Window object*
statusbar, Window.statusbar
StdErr, WScript object*
StdErr, WScript.StdErr
StdIn, WScript object*
StdIn, WScript.StdIn
StdOut, WScript object*
StdOut, WScript.StdOut
Strength, filter - Glow()*
Strength, filter - MotionBlur()*
Strength, filter - Wave()*
stress, style object (2)*
stress, style.stress
StretchStyle, filter - Stretch()*
style, Element object*
style, Element.style
Style, filter - Alpha()*
style, Frame object*
style, rule object*
style, rule.style
styleFloat, style object (2)*
styleFloat, style.styleFloat
SubFolders, Folder object*
Subject, SendMail object*
Subject, SendMail.Subject
suffixes, MimeType object*
summary, TABLE object*
summary, TABLE.summary
sun, Global object*
sun, Window object*
sun, Window.sun
systemId, Entity object*
systemId, Entity.systemId
systemId, Notation object*
systemId, Notation.systemId
systemLanguage, Navigator object*
systemLanguage,
Navigator.systemLanguage
tabIndex, ! object*
tabIndex, A object
tabIndex, Anchor object*
tabIndex, Anchor.tabIndex
tabIndex, Applet object*
tabIndex, Area object*
tabIndex, Area.tabIndex
tabIndex, BODY object*
tabIndex, BODY.tabIndex
tabIndex, BUTTON object*
tabIndex, Embed object*
tabIndex, FIELDSET object*
tabIndex, Form object*
tabIndex, Form.tabIndex
tabIndex, FRAMESET object*
tabIndex, FRAMESET.tabIndex
tabIndex, IFRAME object*
tabIndex, IFRAME.tabIndex
tabIndex, IMG object*
tabIndex, Input object*
tabIndex, Input.tabIndex
tabIndex, Label object*
tabIndex, Legend object*
tabIndex, MARQUEE object*
tabIndex, NOFRAMES object*
tabIndex, NOSCRIPT object*
tabIndex, OBJECT object*
tabIndex, OBJECT.tabIndex
tabIndex, Select object*
tabIndex, TABLE object*
tabIndex, TBODY object*
tabIndex, TD object*
tabIndex, Url object*
tableLayout, style object (2)*
tableLayout, style.tableLayout
tagName, Element object*
tagName, Element.tagName
tagName, Frame object*
tags
tagUrn, Element object*
tagUrn, Element.tagUrn
target, A object
target, Anchor object*
target, Anchor.target
target, Area object*
target, Area.target
target, BASE object*
target, BASE.target
target, Event object*
target, Event.target
target, Form object*
target, Form.target
target, Location object*
target, Location.target

target, Map object*
target, Map.target
target, MouseEvent object*
target, MutationEvent object*
target, ProcessingInstruction object*
target, ProcessingInstruction.target
target, Url object*
target, Url.target
text, Anchor object*
text, Anchor.text
text, Area object*
text, Area.text
text, BODY object*
text, BODY.text
text, COMMENT object*
text, COMMENT.text
text, Location object*
text, Location.text
text, Option object*
text, Option.text
text, SCRIPT object*
text, SCRIPT.text
text, TextRange object*
text, TextRange.text
text, TITLE object*
text, TITLE.text
text, Url object*
text, Url.text
text, XML object*
text, XML.text
textAlign, JSSTag object*
textAlign, JSSTag.textAlign
textAlign, style object (2)*
textAlign, style.textAlign
textAutospace, style object (2)*
textAutospace, style.textAutospace
textDecoration, JSSTag object*
textDecoration, JSSTag.textDecoration
textDecoration, style object (2)*
textDecoration, style.textDecoration
textDecorationBlink, style object (2)*
textDecorationBlink,
style.textDecorationBlink
textDecorationLineThrough, style object (2)*
textDecorationLineThrough,
style.textDecorationLineThrough
textDecorationNone, style object (2)*
textDecorationNone,
style.textDecorationNone
textDecorationOverline, style object (2)*
textDecorationOverline,
style.textDecorationOverline
textDecorationUnderline, style object (2)*
textDecorationUnderline,
style.textDecorationUnderline
textIndent, JSSTag object*
textIndent, JSSTag.textIndent
textIndent, style object (2)*
textIndent, style.textIndent
textJustify, style object (2)*
textJustify, style.textJustify
textKashidaSpace, style object (2)*
textKashidaSpace, style.textKashidaSpace
textShadow, style object (2)*
textShadow, style.textShadow
textTransform, JSSTag object*
textTransform, JSSTag.textTransform
textTransform, style object (2)*
textTransform, style.textTransform
textUnderlinePosition, style object (2)*
textUnderlinePosition,
style.textUnderlinePosition
tFoot, TABLE object*
tFoot, TABLE.tFoot
tHead, TABLE object*
tHead, TABLE.tHead
timeStamp, Event object*
timeStamp, Event.timeStamp
timeStamp, MouseEvent object*
timeStamp, MutationEvent object*
title, Document object*
title, Document.title
title, Element object*
title, Element.title
title, Frame object*
title, HTML object*
title, HTML.title
title, LINK object*
title, LINK.title
title, StyleSheet object*
title, StyleSheet.title
To, SendMail object*
To, SendMail.To
toElement, Event object*
toElement, Event.toElement
toolbar
toolbar, Global object*
toolbar, Window object*
toolbar, Window.toolbar
top
top, Clip object*
top, Clip.top
top, Frame object*
top, Frame.top
top, Global object*
top, JSSTag object*
top, Layer object*
top, Layer.top
top, Rect object*
top, Rect.top
top, style object (2)*
top, style.top
top, textRectangle object*
top, textRectangle.top
top, Window object*
top, Window.top
topMargin, BODY object*
topMargin, BODY.topMargin
TotalSize, Drive object*
TotalSize, Drive.TotalSize
trueSpeed, MARQUEE object*
trueSpeed, MARQUEE.trueSpeed
type, Anchor object*
type, Anchor.type
type, Button object*
type, BUTTON object*
type, Button.type
type, BUTTON.type
type, Checkbox object*
type, Checkbox.type
type, Event object*
type, Event.type
Type, File object*

Property (continued)

Type, File.Type
type, FileUpload object*
type, FileUpload.type
Type, Folder object*
Type, Folder.Type
type, Hidden object*
type, Hidden.type
type, Input object*
type, Input.type
type, LI object*
type, LI.type
type, LINK object*
type, LINK.type
type, MimeType object*
type, MimeType.type
type, MouseEvent object*
type, MutationEvent object*
type, OBJECT object*
type, OBJECT.type
type, OL object*
type, OL.type
type, ParamElement object*
type, ParamElement.type
type, Password object*
type, Password.type
type, RadioButton object*
type, RadioButton.type
type, ResetButton object*
type, ResetButton.type
type, SCRIPT object*
type, SCRIPT.type
type, Select object*
type, Select.type
type, Selection object*
type, selection.type
type, STYLE object (1)*
type, STYLE.type
type, StyleSheet object*
type, StyleSheet.type
type, SubmitButton object*
type, SubmitButton.type
type, TEXTAREA object*
type, TEXTAREA.type
type, TextCell object*
type, TextCell.type
type, UL object*
type, UL.type
type, Url object*
type, Url.type
type, XML object*
type, XML.type
unicodeBidi, style object (2)*
unicodeBidi, style.unicodeBidi
uniqueID, Document object*
uniqueID, Document.uniqueID
uniqueID, Element object*
uniqueID, Element.uniqueID
units, Embed object*
units, Embed.units
updateInterval, Screen object*
updateInterval, Screen.updateInterval
URL, Document object*
URL, Document.URL
url, META object*
url, META.url
urn, Anchor object*
urn, Anchor.urn
urn, Url object*
urn, Url.urn
useMap, IMG object*
useMap, IMG.useMap
useMap, OBJECT object*
useMap, OBJECT.useMap
userAgent, Navigator object*
userAgent, Navigator.userAgent
userLanguage, Navigator object*
userLanguage, Navigator.userLanguage
userProfile, Navigator object*
userProfile, Navigator.userProfile
vAlign, CAPTION object*
vAlign, CAPTION object*
vAlign, CAPTION.vAlign
vAlign, COL object*
vAlign, COL.vAlign
vAlign, COLGROUP object*
vAlign, COLGROUP.vAlign
vAlign, HEAD object*
vAlign, HEAD.vAlign
vAlign, TableColElement object*
vAlign, TableColElement.vAlign
vAlign, TBODY object*
vAlign, TBODY.vAlign
vAlign, TD object*
vAlign, TD.vAlign
vAlign, TFOOT object*
vAlign, TFOOT.vAlign
vAlign, TH object*
vAlign, TH.vAlign
vAlign, THEAD object*
vAlign, THEAD.vAlign
vAlign, TR object*
vAlign, TR.vAlign
value, Attribute object*
value, Attribute.value
value, Button object*
value, BUTTON object*
value, Button.value
value, BUTTON.value
value, Checkbox object*
value, Checkbox.value
value, File object*
value, FileUpload object*
value, FileUpload.value
value, Hidden object*
value, Hidden.value
value, Input object*
value, Input.value
value, LI object*
value, LI.value
value, Option object*
value, Option.value
value, ParamElement object*
value, ParamElement.value
value, Password object*
value, Password.value
value, RadioButton object*
value, RadioButton.value
value, ResetButton object*
value, ResetButton.value
value, Select object*
value, Select.value
value, SubmitButton object*
value, SubmitButton.value

value, TEXTAREA object*
value, TEXTAREA.value
value, TextCell object*
value, TextCell.value
valueType, ParamElement object*
valueType, ParamElement.valueType
version, HTML object*
version, HTML.version
Version, WScript object*
Version, WScript.Version
verticalAlign, JSSTag object*
verticalAlign, JSSTag.verticalAlign
verticalAlign, style object (2)*
verticalAlign, style.verticalAlign
view, MouseEvent object*
view, UIEvent object*
view, UIEvent.view
visibility, JSSTag object*
visibility, Layer object*
visibility, Layer.visibility
visibility, style object (2)*
visibility, style.visibility
visible, Bar object*
visible, Bar.visible
vLink, BODY object*
vLink, BODY.vLink
vlinkColor, Document object*
vlinkColor, Document.vlinkColor
voiceFamily, style object (2)*
voiceFamily, style.voiceFamily
volume, BGSOUND object*
volume, BGSOUND.volume
volume, style object (2)*
volume, style.volume
VolumeName, Drive object*
VolumeName, Drive.VolumeName
vspace, Applet object*
vspace, Applet.vspace
vspace, IFRAME object*
vspace, IFRAME.vspace
vspace, Image object*
vspace, Image.vspace
vspace, IMG object*
vspace, IMG.vspace
vspace, MARQUEE object*
vspace, MARQUEE.vspace
vspace, OBJECT object*
vspace, OBJECT.vspace
which, Event object*
which, Event.which
whiteSpace, JSSTag object*
whiteSpace, JSSTag.whiteSpace
whiteSpace, style object (2)*
whiteSpace, style.whiteSpace
widows, style object (2)*
widows, style.widows
width, Applet object*
width, Applet.width
width, Clip object*
width, Clip.width
width, COL object*
width, COL.width
width, COLGROUP object*
width, COLGROUP.width
width, Document object*
width, Document.width
width, Embed object*
width, Embed.width
width, Event object*
width, Event.width
width, HR object*
width, HR.width
width, IFRAME object*
width, IFRAME.width
width, Image object*
width, Image.width
width, IMG object*
width, IMG.width
width, JSSTag object*
width, JSSTag.width
width, MARQUEE object*
width, MARQUEE.width
width, OBJECT object*
width, OBJECT.width
width, PRE object*
width, PRE.width
width, Rect object*
width, Rect.width
width, Screen object*
width, Screen.width
width, style object (2)*
width, style.width
width, TABLE object*
width, TABLE.width
width, TableColElement object*
width, TableColElement.width
width, TD object*
width, TD.width
width, TH object*
width, TH.width
window
window, Global object*
window, Layer object*
window, Layer.window
window, Window object*
window, Window.window
WipeStyle, filter - GradientWipe()*
WipeStyle, filter - RadialWipe()*
wordBreak, style object (2)*
wordBreak, style.wordBreak
wordSpacing, style object (2)*
wordSpacing, style.wordSpacing
wordWrap, style object (2)*
wordWrap, style.wordWrap
wrap, TEXTAREA object*
wrap, TEXTAREA.wrap
writingMode, style object (2)*
writingMode, style.writingMode
x, Anchor object*
x, Anchor.x
x, Area object*
x, Area.x
x, Event object*
x, Event.x
x, Image object*
x, Image.x
x, Layer object*
x, Layer.x
x, Location object*
x, Location.x
x, Url object*
x, Url.x
XMLDocument, XML.XMLDocument
XRay, filter - BasicImage()*

Property (continued)
y, Anchor object*
y, Anchor.y
y, Area object*
y, Area.y
y, Event object*
y, Event.y
y, Image object*
y, Image.y
y, Layer object*
y, Layer.y
y, Location object*
y, Location.y
y, Url object*
y, Url.y
zIndex, JSSTag object*
zIndex, Layer object*
zIndex, Layer.zIndex
zIndex, style object (2)*
zIndex, style.zIndex
zoom, style object (2)*
zoom, style.zoom

Property attribute
DontDelete*
DontEnumerate*
ReadOnly*

Property/internal
Array.Class*
Boolean.Class*
Class*
Construct*
Date.Class*
Function.Class*
Image.Class*
Number.Class*
Object.Class*
String.Class*

Property/static
$n (Numbered argument)
$n, RegExp.$n*
input, RegExp.input*
lastMatch, RegExp.lastMatch*
lastParent, RegExp.lastParent*
leftContext, RegExp.leftContext*
multiline, RegExp.multiline*
rightContext, RegExp.rightContext*
RegExp["$'"]
RegExp["$*"]
RegExp["$`"]
RegExp["$+"]

Request method
about: URL*
clsid: URL*
file: URL*
ftp: URL*
http: URL*
https: URL*
JavaScript interactive URL*
javascript: URL*
livescript: URL
mailbox: URL*
mailto: URL
mocha: URL
nethelp: URL*
news: URL*
snews: URL*
telnet: URL*
view-source: URL*
wysiwyg:*

Reserved word
abstract
boolean*
byte*
char*
class*
const*
debugger
double*
enum*
extends
final
float*
goto*
implements
int*
interface
long*
native
package
private
protected
public
short*
static
super
synchronized
throws
transient
volatile*

Security privilege
UniversalBrowserAccess*
UniversalBrowserRead*
UniversalBrowserWrite*
UniversalFileRead*
UniversalPreferencesRead*
UniversalPreferencesWrite*
UniversalSendMail*

Security related
AuthentiCode*
Cryptoki*
Data-tainting*
Requesting privileges*
Same origin*
Signed scripts*

Selector (see also Label)
else if(...) ...
if(...) ... else ...*
if(...) ...*
switch(...) ... case: ... default: ...*

Simulated functionality
isAlnum()
isAlpha()
isCtrl()
isDigit()
isElementProperty()
isGraph()
isLower()
isObjectEqual()
isODigit()
isPrint()
isPunct()
isSpace()
isUpper()
isXDigit()
Math.cosec()
Math.cosh()
Math.cot()
Math.sec()
Math.sinh()

Special file (see also File extension)
config.jsc
preferences.js
prefs.js
proxy.pac*

Standard (see also Background, Definition, Overview)
ASCII*
ATVEF*
CSS level 1*
CSS level 2*
CSS*
CSS-P*
DHTML*
DOM - Level 0*
DOM - Level 1*
DOM - Level 2*
DOM - Level 3*
DOM Events*
DOM*
DVB-MHP*
ECMA*
ECMAScript - edition 2*
ECMAScript - edition 3*
ECMAScript version*
HTML*
IEEE 754*
ISO 3166*
ISO 639
JavaScript version*
JScript version*
PDF*
Unicode*
Universal coordinated time*
UTC
WAP*
WML*
WScript*
XML*

Statement
Block { }
break*
continue*
Empty statement (;)*
export*
finally ...*
import*
return*
throw*
try ... catch ... finally*
with ...*

Time calculation
Date from time
Date number
Day from year
Day number
Day within year
Days in year
In leap year
MakeDate()
MakeDay()
MakeTime()
Month from time
Month number
Time from year
Time value
Time within day
TimeClip()
Week day
Year from time
Year number

Type
Boolean*
null*
Number*
Object*
String*
undefined type*

Useful tip (see also Advice, Pitfall)
Determining the object type
Image animation
Image preloading
Object inspector
Off-screen image caching
Queue manipulation
Server-side browser detection
Stack manipulation
Static variable

Web browser
iCab*
Internet Explorer*
MSIE
Netscape Navigator*
Opera*